



---

# LINGUAGEM E PADRÕES WEB

**Anderson da Silva Marcolino**

# **LINGUAGEM E PADRÕES WEB**

**1<sup>a</sup> edição**

Londrina  
Editora e Distribuidora Educacional S.A.  
2021

**© 2021 por Editora e Distribuidora Educacional S.A.**

Todos os direitos reservados. Nenhuma parte desta publicação poderá ser reproduzida ou transmitida de qualquer modo ou por qualquer outro meio, eletrônico ou mecânico, incluindo fotocópia, gravação ou qualquer outro tipo de sistema de armazenamento e transmissão de informação, sem prévia autorização, por escrito, da Editora e Distribuidora Educacional S.A.

**Presidente**

Rodrigo Galindo

**Vice-Presidente de Pós-Graduação e Educação Continuada**

Paulo de Tarso Pires de Moraes

**Conselho Acadêmico**

Carlos Roberto Pagani Junior  
Camila Braga de Oliveira Higa  
Carolina Yaly  
Giani Vendramel de Oliveira  
Gislaine Denisale Ferreira  
Henrique Salustiano Silva  
Mariana Gerardi Mello  
Nirse Ruscheinsky Breternitz  
Priscila Pereira Silva  
Tayra Carolina Nascimento Aleixo

**Coordenador**

Henrique Salustiano Silva

**Revisor**

Gabriela Silveira Barbosa

**Editorial**

Alessandra Cristina Fahl  
Beatriz Meloni Montefusco  
Gilvânia Honório dos Santos  
Mariana de Campos Barroso  
Paola Andressa Machado Leal

Dados Internacionais de Catalogação na Publicação (CIP)

---

Marcolino, Anderson da Silva  
M321 Linguagem e Padrões Web / Anderson da Silva  
Marcolino, – Londrina: Editora e Distribuidora Educacional  
S.A., 2021.  
44 p.

ISBN 978-65-5903-110-8

1. HTML. 2. CSS. 3. Javascript. I. Título.

CDD 003

---

Evelyn Moraes – CRB 010289/O

2021

Editora e Distribuidora Educacional S.A.  
Avenida Paris, 675 – Parque Residencial João Piza  
CEP: 86041-100 — Londrina — PR  
e-mail: editora.educacional@kroton.com.br  
Homepage: <http://www.kroton.com.br/>

## LINGUAGEM E PADRÕES WEB

### SUMÁRIO

Dominando a linguagem de marcação de hipertexto – HTML	05
Estilização das páginas Web: as folhas de estilo em cascata	26
Linguagem JavaScript: do básico ao avançado	42
Implementação de Páginas Web com HTML, CSS e JavaScript	61

# Dominando a linguagem de marcação de hipertexto – HTML.

Autoria: Anderson da Silva Marcolino

Leitura crítica: Gabriela Silveira Barbosa



## Objetivos

- Conhecer o contexto histórico da linguagem de marcação de hipertexto (HTML) e a evolução de suas versões.
- Identificar as diferenças entre HTML e linguagem de marcação de hipertexto extensível (XHTML) e linguagem de marcação extensível (XML).
- Compreender e utilizar as principais marcações da linguagem HTML, essenciais para a criação de uma página HTML.

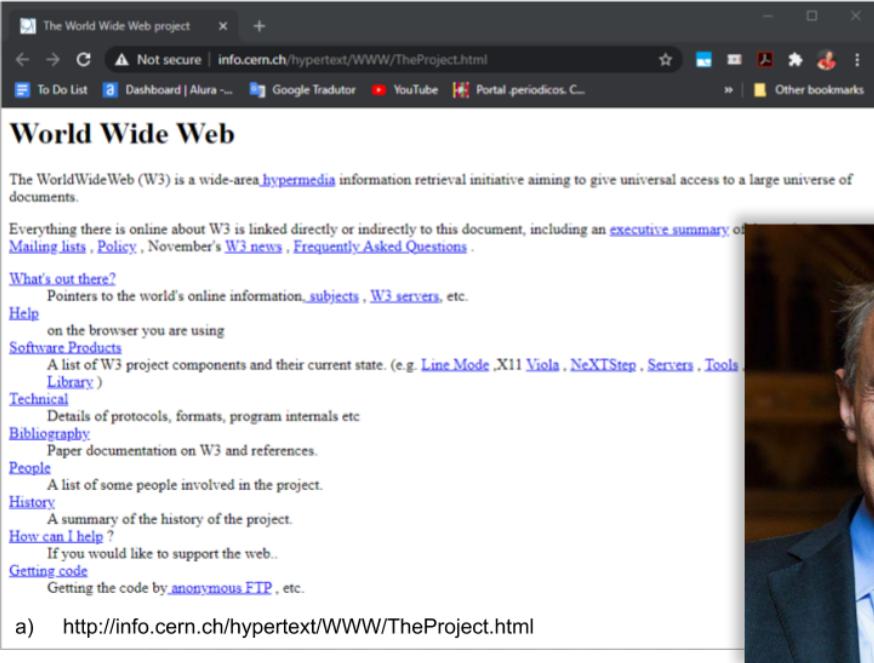


# 1. Introdução

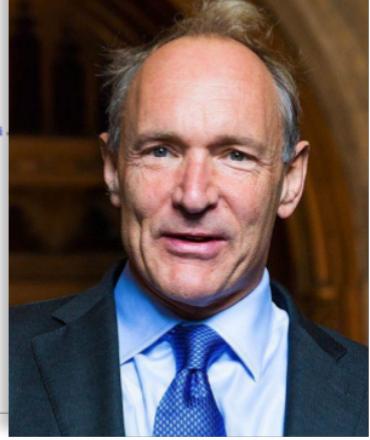
O surgimento da Internet, na década de 1960, levou a uma nova era de avanços tecnológicos e expansão do mercado de tecnologia, principalmente, no contexto de desenvolvimento web.

O primeiro website foi desenvolvido no Centro Europeu de Pesquisa Nuclear (CERN), publicado no dia seis de agosto de 1991. A ideia inicial veio do físico Tim Berners-Lee (Figura 1 b) que escreveu uma proposta mostrando como a informação poderia ser facilmente divulgada e conectada por meio de hiperlinks. Seu projeto foi intitulado WorldWideWeb, e sua interface pode ser visualizada na Figura 1, em a), segundo Laura (2019).

**Figura 1 – a) Primeiro website e b) Foto do Tim Berners-Lee**



The screenshot shows a Windows-style desktop browser window. The title bar says 'The World Wide Web project'. The address bar shows 'Not secure | info.cern.ch/hypertext/WWW/TheProject.html'. Below the address bar is a toolbar with icons for 'To Do List', 'Dashboard | Alura ...', 'Google Tradutor', 'YouTube', 'Portal periódicos. C...', and 'Other bookmarks'. The main content area displays the 'World Wide Web' page. The page text reads: 'The WorldWideWeb (W3) is a wide-area [hypermedia](#) information retrieval initiative aiming to give universal access to a large universe of documents.' It also lists links to 'Mailing lists', 'Policy', 'November's [W3 news](#)', and 'Frequently Asked Questions'. On the left, there's a sidebar with links like 'What's out there?', 'Help', 'Software Products', 'Technical', 'Bibliography', 'People', 'History', 'How can I help?', and 'Getting code'. At the bottom, it shows the URL 'a) http://info.cern.ch/hypertext/WWW/TheProject.html'.



b) Tim Berners-Lee

Fonte: a) captura de tela. b) [https://commons.wikimedia.org/wiki/File:Sir\\_Tim\\_Berners-Lee\\_\(cropped\).jpg](https://commons.wikimedia.org/wiki/File:Sir_Tim_Berners-Lee_(cropped).jpg). Acesso em: 5 mar. 2021.

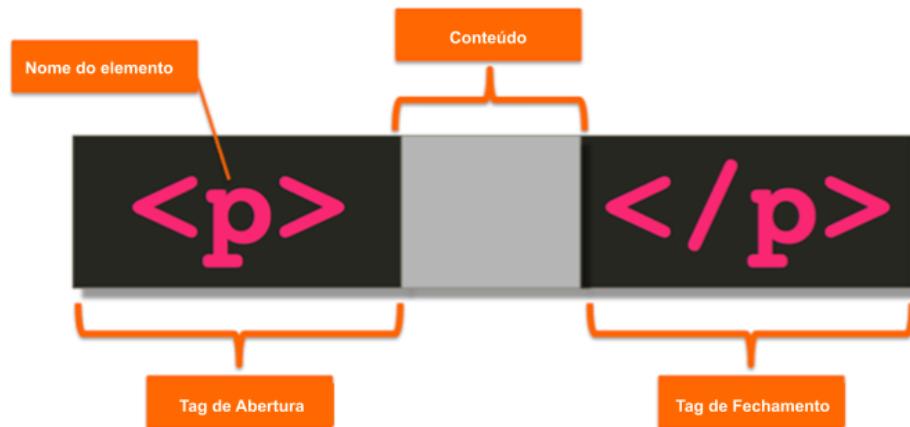
O website presente na Figura 1, em a), foi criado com a linguagem de marcação de hipertexto (HTML) puro. Isso ocorreu por ser ainda

o começo da difusão da Internet, que só veio a se popularizar como surgimento do NCSA Mosaic, um navegador gráfico criado, inicialmente, para Unix, mas que foi adaptado também para os sistemas operacionais da Apple Macintosh e Microsoft Windows. A primeira versão do Mosaic foi lançada em setembro de 1993 (EIS; FERREIRA, 2012; FREEMAN, 2008). Desde então, tecnologias utilizadas para o desenvolvimento de páginas web, ou websites, tem apresentado grandes inovações. Trazendo, mais recentemente, o surgimento de aplicações web.

Um website pode ser considerado um conjunto de páginas estáticas, desenvolvidas com HTML, que não possuem conexão com bases de dados ou com linguagens desenvolvidas e mantidas em servidores web. Já uma aplicação web, contém, além das páginas estáticas em HTML(também são chamadas de documentos HTML), meios de comunicação com banco de dados, além de integrar e utilizar linguagens de programação que interagem com um ou mais servidores web. A parte que o usuário interage, no contexto das aplicações web, é chamada de *front-end*. Já a parte mantida nos servidores, e consumida por meio de requisições dos clientes, é denominada *back-end* (EIS; FERREIRA, 2012).

É nítida a importância de HTML para o desenvolvimento tanto de websites, quanto de aplicações web. Contudo, é necessário esclarecer que HTML, que hoje se encontra em sua versão 5.2, não é uma linguagem de programação, mas uma linguagem de marcação. Essa linguagem usa marcações, ou *tags*, em inglês, que, em sua maioria, apresentam um elemento textual simbólico de abertura e outro de fechamento, formando um elemento web, como pode ser apresentado na Figura 2. Suas versões e padronizações são mantidas pela *World Wide Web Consortium*, que é a principal organização de padronização da *World Wide Web*.

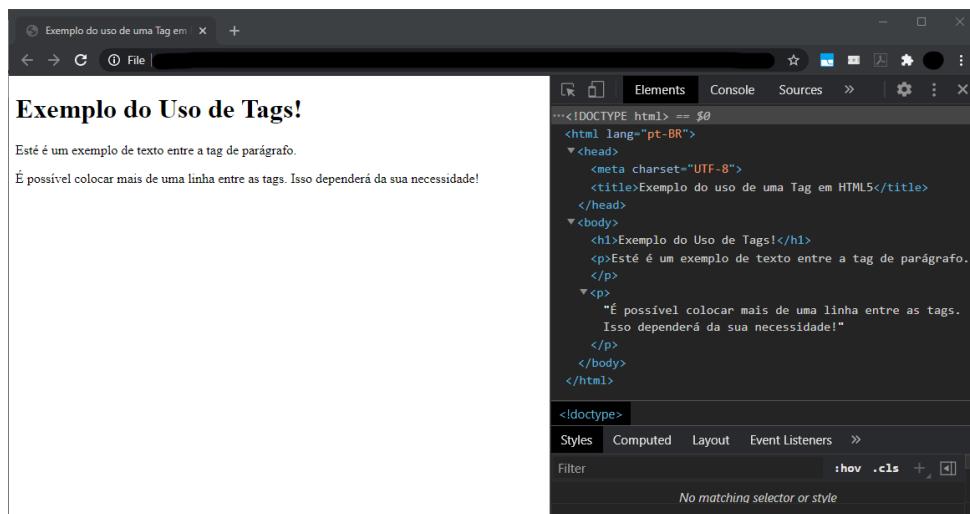
## Figura 2 – Sintaxe de uma tag HTML



Fonte: adaptado de CHAIKIN (2017).

Na Figura 2, temos o elemento de parágrafo. É uma *tag* textual, que formatará o texto inserido nela como um parágrafo, um espaço em branco na primeira linha, bem como uma quebra de linha antes deste espaço. Essa formatação ocorre pelo navegador quando encontra tal marcação no documento HTML do website. Quando uma nova versão do HTML é lançada, muitos navegadores necessitam de atualização para suportar a nova versão. A Figura 3 mostra um exemplo de código HTML, utilizando a *tag* de parágrafo e o resultado obtido no navegar Google Chrome.

## Figura 3 – Exemplo de página HTML e uso da tag parágrafo



Fonte: print de tela do Navegador Google Chrome v88.0

Na Figura 3, temos um primeiro exemplo completo de uma página HTML5 e a estrutura essencial para desenvolver um website. Primeiramente, concentraremos no uso do elemento *Parágrafo*.

Na primeira linha, temos a declaração `<!DOCTYPE>`, que deve ser escrita sempre na primeira linha, pois será utilizada pelo navegador para renderizar, ou seja, criar e exibir a página HTML de acordo com a versão HTML indicada na sequência desta palavra. A declaração `<!DOCTYPE>` não é uma *tag* HTML, mas uma instrução para o navegador.

A seguir, temos a *tag* “`<html lang="pt-BR">`”, que tem seu fechamento na última linha, pela *tag* `</html>`, que delimita o documento HTML. O atributo *lang* é opcional e indica qual a linguagem do documento HTML, no caso, português do Brasil. Isso ajuda o navegador a fornecer suporte a tradução ou não da página, dependendo das configurações regionais definidas tanto no navegador, quanto no computador do usuário. É importante destacar que várias *tags* do HTML possuem atributos próprios, ou seja, só podem ser utilizadas por elas; outras *tags* usam atributos globais, que várias *tags* podem utilizar; e a de eventos, utilizada especialmente para chamada de funções e códigos em JavaScript. Todos os atributos das *tags* são definidos no formato *chave="valor"* e dentro da *tag* de abertura, como no exemplo da *tag* `<html lang="pt-BR">`.

Na sequência, temos a seção de cabeçalho, delimitada pelas *tags* `<head></head>`. O conteúdo inserido entre este elemento resume-se a dados para otimização de busca, tipo de caracteres utilizado (definido na Figura 3 por “`<meta charset="UTF-8">`”) e a *tag* de título, que exibirá o título ali inserido, na barra de títulos do navegador, como também pode ser visto na Figura 3.

O último elemento essencial para a criação de um website é o elemento `<body>`, ou seja, o *corpo*, na tradução do inglês para o português. É nesse elemento que as *tags* são inseridas para gerar a página web final.

Assim, no corpo do documento, temos o elemento parágrafo, utilizado duas vezes. No primeiro uso, as *tags* estão entre uma única linha de texto. Enquanto no segundo, entre duas. Entretanto, o resultado no navegador é o mesmo. Ambos são marcados como parágrafos.

No mesmo documento, temos o uso de uma outra *tag*, detítulo *<h1>*. Tal como a *tag* de parágrafo, esta possui *tag* de abertura e fechamento. O texto inserido entre as *tags* de tal elemento ficará em uma fonte maior e em negrito, podendo ainda variar o tamanho do texto, bastando, para isso, substituir o número um por números que vão até seis. Por exemplo, “*<h2>Texto qualquer!</h2>*”, e assim por diante.

Com base nessas especificações, diferenciaremos HTML de outras linguagens de marcação e identificaremos as principais *tags* da linguagem, a fim de permitir o desenvolvimento tanto de websites, quanto de aplicações web.

## 1.1 Diferenças entre HTML, XML e XHTML

Como HTML é uma linguagem de marcação específica para o desenvolvimento de páginas web, podemos identificar similaridades e também diferenças entre outras linguagens de marcação de igual importância no contexto de desenvolvimento de software em geral, como XML e XHTML.

O *Extensible Markup Language* (XML), ou seja, linguagem de marcação extensível é uma linguagem que define regra para codificação de documentos. Entretanto, ao contrário de HTML, que é uma linguagem especializada para a criação de páginas web, XML é genérica, permitindo a criação de *tags* específicas. Em outras palavras, fornece meio para definir elementos de marcação para gerar uma linguagem personalizada, segundo Freeman (2008).

Um arquivo XML, é dividido em duas partes: *prolog*, que consiste em metadados administrativos, assim como a maioria dos itens inseridos no cabeçalho (“*<head>*”) de um documento HTML; e o *body*, que integra a parte estrutural e conteúdo, similar ao HTML.

As marcações XML sempre são compostas por uma *tag* de abertura e de fechamento, diferentemente de HTML, que pode possuir *tags* que não possuem fechamento, chamadas de *tags self close*, ou seja, que fecham por si só, não precisam de uma *tag* de fechamento. Por exemplo, a *tag* *<br>* de quebra de linha e a *<hr>* para inserir uma linha horizontal. Contudo, no contexto da versão 5 do HTML, recomenda-se utilizar a barra “/” para fechar também *tags self close*.

Finalmente, um exemplo de uso cotidiano do XML, é a representação de dados, como os utilizados nos sistemas de notas fiscais eletrônicas.

O XHTML é a linguagem de marcação extensível de hipertexto. Faz parte da família das linguagens XML, que estende as versões do HTML. Enquanto HTML é uma linguagem para formato de arquivo de documento, XHTML é considerado uma linguagem de marcação, de fato. Isso significa que XHTML integra a padronização mais formal do XML, que exige sempre a abertura e fechamento de *tags* com a integração das marcações definidas no HTML, segundo Freeman (2008).

Retomando os exemplos mencionados das *tags self close*, como *<br>* e *<hr>* do HTML. Ao usar as mesmas *tags* em XHTML, haverá a necessidade de tais *tags* apresentarem tanto a abertura, como o fechamento: “*<br> <br/>*”. O HTML5 integra fortemente a padronização das *tags* que possuem abertura e fechamento. Contudo, muitos navegadores responsáveis por ler o documento HTML solicitado pelos clientes, por meio de um endereço de um recurso na rede via URL (*Uniform Resource Locator*), que, em português, significa *Localizador Padrão de Recursos*, inserem fechamentos para *tags*, evitando erros e exibindo a página HTML como esperado.

Finalmente, um arquivo HTML deve ser salvo com tal extensão, por exemplo, *index.html*. Por padrão, para que o navegador saiba qual a página inicial de um website, deve ser salvo como *index*, que significa índice.

Agora que sabemos como utilizar as *tags* em um documento HTML e as diferenças entre tal documento com os arquivos XML e XHTML. Vejamos os principais elementos a serem utilizados na criação de websites.

## 1.2 Principais elementos HTML por categoria

Podemos dividir os elementos HTML em diversas categorias, sendo as principais (W3C, 2017; W3Schools, 2020):

- HTML básico.
- Formatação.
- Formulários.
- Imagens, áudio/ vídeo.
- Links.
- Listas.
- Tabelas.
- Estilos e semântica.
- Meta informação.
- Programação.

Vejamos o conjunto principal de *tags* dessas categorias, sua descrição e exemplos de uso, nos quadros que seguem.

## Quadro 1 – Elementos essenciais básicos do HTML5

<b>Tag</b>	<b>Descrição</b>	<b>Exemplo</b>
<!DOCTYPE>	Define o tipo de documento.	<!DOCTYPE HTML 4>
<html>	Define um <i>document</i> HTML.	<!DOCTYPE html>
<head>	Seção do documento HTML, que contém metadados e informações do documento. Não inclui conteúdos das páginas.	<html lang="pt-BR">  <head>  <title>Título do Documento</title>  </head>
<title>	Define o título do documento.	<body>
<body>	Define o corpo do documento. É onde serão incluídas as <i>tags</i> de conteúdo da página.	<h1>Este é um Título</h1>  <p>Um parágrafo qualquer.</p>  </body>  </html>
<h1> até <h6>	Define os títulos de um documento HTML, com tamanhos e formatações diferentes para cada número indicado na <i>tag</i> .	<h1></h1>
<p>	Define um parágrafo no documento HTML.	<p>Exemplo de parágrafo!</p>

 	Insere uma quebra de linha no documento. É uma <i>tag self close</i> .	 
<hr>	Insere uma linha ou divisão de conteúdo, como uma quebra de seção. É uma <i>tag self close</i> .	<hr>
<!-- ... -->	Define um bloco de comentário no documento, não sendo exibido na página web.	<!--Exemplo de comentário no document HTML -->

Fonte: adaptado e traduzido de W3Schools (2020).

## Quadro 2 – Elementos essenciais de formatação, formulários, mídias, links e listas do HTML5

<b>Formatação</b>		
<b>Tag</b>	<b>Descrição</b>	<b>Exemplo</b>
<em>	Define textos em itálico. Texto com ênfase.	<em>Texto com ênfase!</em>
<strong>	Define textos em negrito. Texto importante.	<strong>Texto importante!</strong>
<b>Formulários</b>		
<form>	Define uma estrutura de formulário, para incluir <i>inputs</i> e demais elementos da lista de formulários.	<form></form>

<input>	Define um controle de entrada de dados ( <i>input</i> ).	<pre>&lt;form action="/pagina_acao.php"&gt;  &lt;label for="fnome"&gt;Primeir nome:&lt;/label&gt;  &lt;input type="text" id="fnome" name="fnome"&gt;&lt;br&gt;  &lt;input type="submit" value="Submit"&gt;  &lt;/form&gt;</pre>
<label>	Define um rótulo para um elemento de entrada de dados ( <i>input</i> , <i>textarea</i> e outros tipos de campos).	<pre>&lt;form action="/pagina_acao.php"&gt;  &lt;label for="masculino"&gt; Masculino &lt;/label&gt;  &lt;input type="radio" name="sexo" id=" masculino " value=" masculino "&gt;&lt;br&gt;  &lt;label for="feminino"&gt; Feminino &lt;/label&gt;  &lt;input type="radio" name="sexo" id="female" value=" feminino "&gt;&lt;br&gt;  &lt;input type="submit" value="Submit"&gt;  &lt;/form&gt;</pre>

<textarea>	Define um campo de entrada com várias linhas, ou seja, com a quantidade de linhas ( <i>rows</i> ) e colunas ( <i>cols</i> ) configuráveis.	<label for="mensagemtext">Mensagem:</label>  <textarea id="mensagemtext" name="mensagemtext" rows="4" cols="50"> </textarea>
<button>	Define um botão que o usuário pode clicar.	<button type="button">Texto do Botão!</button>
<b>Imagens, áudio/ vídeo</b>		
<img>	Define uma imagem. É uma <i>tag self close</i> .	
<audio>	Define um conteúdo de som, cujo os elementos nele inseridos utilizam a <i>tag &lt;source&gt;</i> .	<audio controls>  <source src="som1.ogg" type="audio/ogg">  <source src="som2.mp3" type="audio/mpeg">  Seu navegador não suporta a <i>tag</i> de áudio.  </audio>
<picture>	Define um contêiner para múltiplas imagens, que podem ser definidas por meio das <i>tag &lt;img&gt;</i> ou <i>&lt;source&gt;</i> .	<picture>  <source media="(min-width:650px)" srcset="img_flores.jpg">    </picture>

<video>	Define um elemento de vídeo ou filme, sendo cada um dos elementos, nele inseridos, definidos por meio da tag <source>.	<pre>&lt;video width="320" height="240" controls&gt;  &lt;source src="filme1.mp4" type="video/mp4"&gt;  &lt;source src="video2.ogg" type="video/ogg"&gt;  Seu navegador não suporta a tag de vídeo.  &lt;/video&gt;</pre>
<source>	Define recursos de mídia para múltiplas fontes (<video>, <audio> and <picture>). É uma tag <i>self close</i> .	<pre>&lt;source src="musica.mp3" type="audio/mpeg"&gt;</pre>
<b>Links</b>		
<a>	Define um hiperlink ou hiperligação. É um elemento que, ao ser clicado pelo usuário, leva a uma outra página ou conteúdo na Internet. É o termo que dá origem ao H na sigla HTML.	<pre>&lt;a href="https://www.w3schools.com"&gt;Visite W3Schools.com!&lt;/a&gt;</pre>
<link>	Define uma relação entre um documento HTML e um recurso externo, como uma folha em estilos em cascata (arquivo de extensão .css). É uma tag <i>self close</i> .	<pre>&lt;link rel="stylesheet" href="styles.css"&gt;</pre>

<nav>	Define links de navegação, ou seja, concentra mais de um elemento identificado pela tag <a>, formando um menu.	<nav>  <a href="/html/">HTML</a>    <a href="/css/">CSS</a>    </nav>
<b>Listas</b>		
<ul>	Define uma lista de itens não ordenados, ou seja, que não seguem uma ordem numérica ou alfabética.	<ul>  <li>Café</li>  <li>Leite</li>  </ul>
<ol>	Define uma lista de itens ordenados. Pode usar o atributo start="50" para definir o início. No exemplo, a lista iniciará no número 50.	<ol start="50">  <li>Café</li>  <li>Achocolatado</li>  </ol>
<li>	Define um item de uma lista ordenada ou não.	<li>Café</li>

Fonte: adaptado e traduzido de W3Schools (2020).

### Quadro 3 – Elementos essenciais de tabelas, estilos e semântica, meta informação e programação do HTML5

Tabelas		
Tag	Descrição	Exemplo
<table>	Define uma tabela. Pouco usado, após a criação da tag de <div>.	<table>  <tr>  <th>Mês</th>  <th>Lucros</th>  </tr>  <tr>  <td>Janeiro</td>  <td>R\$ 100</td>  </tr>  </table>
<th>	Define a célula de cabeçalho em uma tabela.	
<tr>	Define uma linha em um tabela.	
<td>	Define uma célula de uma tabela.	

<b>Estilos e Semântica</b>		
<style>	<p>Define informações de estilos para um documento e seus elementos. É inserido no elemento de cabeçalho do documento HTML (&lt;head&gt;).</p>	<pre>&lt;html&gt;   &lt;head&gt;     &lt;style&gt;       h1 {color:red;}       p {color:blue;}     &lt;/style&gt;   &lt;/head&gt;   &lt;body&gt;     &lt;h1&gt;Um cabeçalho.&lt;/h1&gt;     &lt;p&gt;Um parágrafo.&lt;/p&gt;   &lt;/body&gt; &lt;/html&gt;</pre>

<div>	<p>Define uma seção em um documento.</p> <p>Muito utilizado para criar quadros e tabelas com a integração de folhas de estilo em cascata (CSS) ou a aplicação de estilhos, como no exemplo ao lado. O atributo class="myDiv" é usado para criar um seletor de estilos. É por meio deste atributo que o navegador sabe qual estilo, dentro de uma lista de estilos na tag &lt;style&gt; que deve aplicar.</p>	<pre>&lt;html&gt; &lt;head&gt; &lt;style&gt;  .myDiv {     border: 5px outset red;     background-color: lightblue;     text-align: center; }  &lt;/style&gt; &lt;/head&gt; &lt;body&gt; &lt;div class="myDiv"&gt;     &lt;h2&gt;Cabeçalho dentro de uma DIV&lt;/h2&gt;     &lt;p&gt;Texto dentro de uma DIV.&lt;/p&gt; &lt;/div&gt; &lt;/body&gt; &lt;/html&gt;</pre>
-------	--	--

<span>	Define uma seção no documento. Muito utilizada para alterar estilos de textos.	<p>Minha mãe tem olhos <span style="color:blue">azuis</span>.</p>
<article>	Define um artigo que deve estar inserido no elemento <body> de um documento HTML.	[...]  <article>  <header>  <h1>Cabeçalho</h1>  <p>Postado por Alguém</p>  </header>
<header>	Define um cabeçalho do artigo (<article>).	<p>Lorem ipsum dolor set amet....</p>  </article>
<footer>	Define um rodapé para uma página ou artigo.	<footer>  <p>Author: Algueém</p>  <p><a href="mailto:alguem@gmail.com">Contato</a></p>  </footer>  [...]

Meta informação		
<meta>	Define metadados a respeito de um arquivo HTML. Muito utilizado para definir informações úteis para mecanismos de busca. Sempre será inserido no elemento <head>.	[...]  <head>  <meta charset="UTF-8">  <meta name="description" content="Descrição qualquer.">  <meta name="keywords" content="Palavras Chame">  <meta name="author" content="Nome do Autor">  <meta name="viewport" content="width=device-width, initial-scale=1.0"><!-- Para iniciar na escala da tela do dispositivo! Sempre inserir! -->  </head>  [...]

Programação		
<script>	Define um arquivo de <i>script</i> a ser interpretado do lado do cliente como, por exemplo, um comando em JavaScript que imprime algo, como no exemplo ao lado.	<pre>&lt;script&gt; document.getElementById("demo").innerHTML = "Hello JavaScript!"; &lt;/script&gt;</pre>
<noscript>	Define um conteúdo alternativo a ser exibido para usuários, quando o navegador não suportar <i>scripts</i> .	<pre>&lt;script&gt; document.write("Hello World!") &lt;/script&gt;  &lt;noscript&gt;Seu navegador não suporta JavaScript!&lt;/noscript&gt;</pre>

Fonte: adaptado e traduzido de W3Schools (2020).

É possível perceber que os quadros apresentados trazem uma diversidade de elementos HTML a serem utilizados. A melhor forma de entender ainda mais cada uma das marcações, definidas em HTML, é criar um arquivo HTML. Para isso, você pode testar os códigos criando um arquivo no bloco de notas ou qualquer editor simples de texto e depois salvar o arquivo como *.html*. Ao procurar o arquivo para abrir, este será automaticamente aberto no navegador e exibirá o resultado do arquivo HTML renderizado, ou seja, criado pelo navegador de sua escolha. Utilize as marcações e integre-as em diversos projetos

diferentes. Caso tenha dificuldades em criar o primeiro arquivo *html*, utilize o *index.html* disponibilizado como material extra deste tema!

Bons estudos!

## Referências

EIS, D.; FERREIRA, E. **HTML5 e CSS3 com farinha e pimenta.** São Paulo: Tableless, 2012.

FREEMAN, E. **Use a cabeça!:** HTML com CSS e XHTML. 2. ed. Rio de Janeiro: Alta books, 2008.

CHAIKIN, Y. **Anatomy of an HTML Tag.** 2017. Disponível em: <https://clearlydecoded.com/anatomy-of-html-tag>. Acesso em: 9 mar. 2021.

LAURA, A. G. **XHTML/CSS:** criação de páginas web. São Paulo: Senac, 2019.

WORLD WIDE WEB CONSORTIUM (W3C). **HTML 5.2.** 2017. Disponível em: <https://www.w3.org/TR/html52/>. Acesso em: 5 mar. 2021.

W3SCHOOLS. **HTML Reference.** 2020. Disponível em: <https://www.w3schools.com/tags/default.asp>. Acesso em: 5 mar. 2021.

# Estilização das páginas Web: as folhas de estilo em cascata.

Autoria: Anderson da Silva Marcolino

Leitura crítica: Gabriela Silveira



## Objetivos

- Conhecer o contexto histórico da estilização das páginas em linguagem de marcação de hipertexto (HTML) e a utilização das folhas de estilo em cascata.
- Apresentar a gramática, convenções, herança e escopo das estruturas das folhas de estilo.
- Conhecer e utilizar o *Box Model* e compreender a responsividade na web.

## 1. Introdução

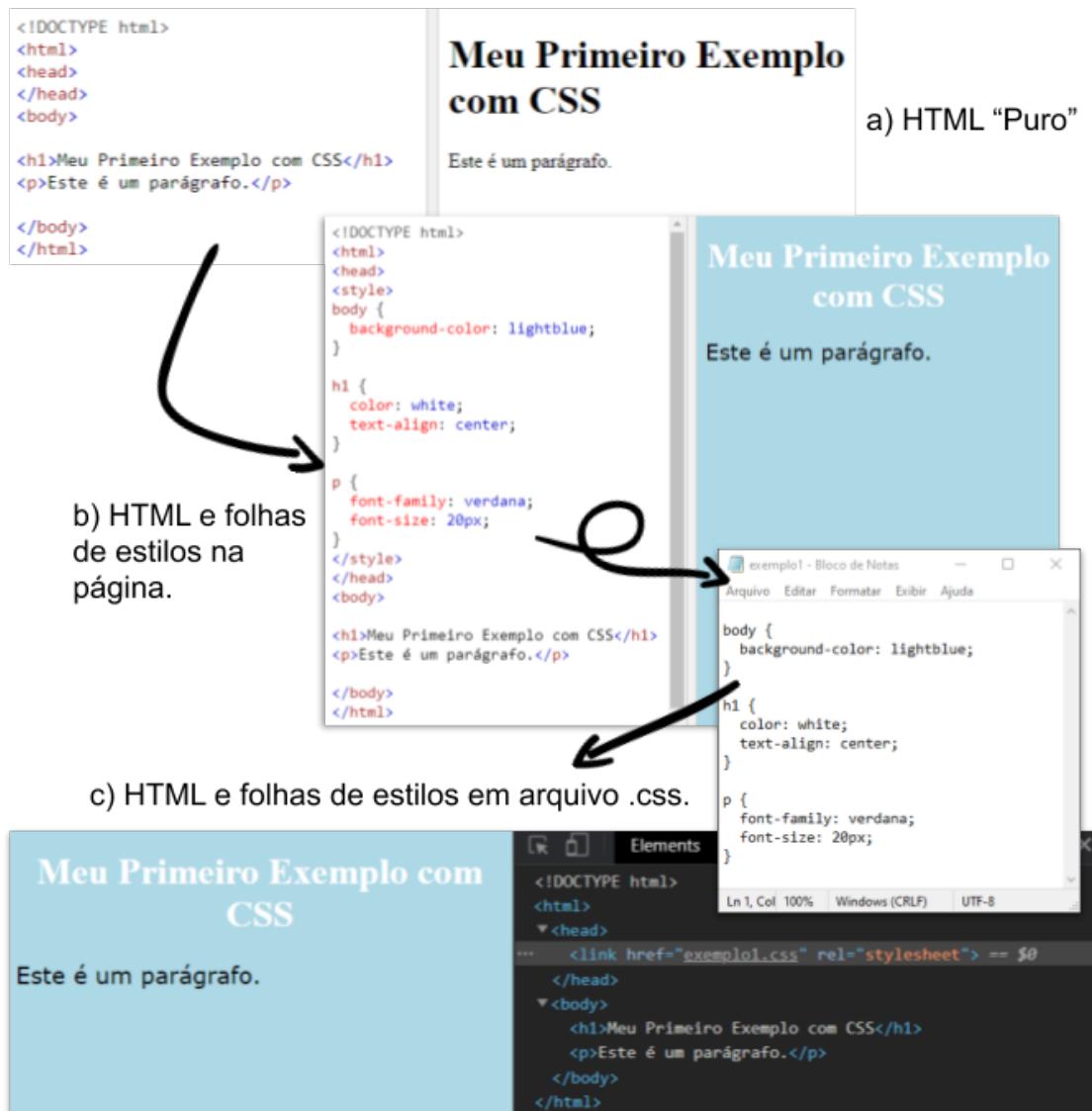
O diferencial de websites e aplicações web são suas interfaces e elementos gráficos que tornam a página atrativa, fazendo com que os usuários permaneçam ali conectados, consumindo informações ou utilizando o site para compras etc. Contudo, o HTML não consegue, sozinho, criar tais websites atrativos.

É necessário integrar estilos aos arquivos. Estes estilos podem ser inseridos na seção de cabeçalho de uma página HTML, por meio da *tag* específica, como, por exemplo: "<style> h1 {color:red;} p {color:blue;} </style>". Nesse exemplo, um estilo é aplicado às *tags* *h1* e *p*, cuja propriedade *color* recebe como valor a cor vermelha (*red*) e azul (*blue*), respectivamente. Todo elemento HTML recebe por padrão um estilo próprio, definido pelo próprio navegador. Os estilos, quando modificados por meio de propriedades e valores de propriedades (*color: blue*) alteram a aparência do elemento HTML, que é selecionado por meio de um seletor. No caso, os seletores utilizados no exemplo representam o próprio nome da *tag* do HTML: *h1* e *p*.

Os estilos, quando aplicados em contexto mais profissional, indo além do aprendizado inicial, ocorrem por meio de arquivos independentes, com extensão ".css" inseridos também no cabeçalho de uma página. Entretanto, por meio da *tag* *link*: "<link href="local\_do\_arquivo/estilo.css" rel="stylesheet">". Nessa tag, temos o *href* referenciando o local onde o arquivo .css será encontrado, com o nome da pasta e o próprio nome do arquivo. Já o *rel* indica o tipo de conteúdo que está relacionado ao link, no caso, *stylesheet*, ou folha de estilo.

A Figura 1 apresenta uma página HTML pura (Figura 1, a), uma página HTML com estilos aplicados por meio da *tag* *<style>* (Figura 1, b) e outra, com estilos aplicados por meio de um arquivo .css (Figura 1, c).

**Figura 1 – Uma página HTML com: a) sem CSS; b) com CSS no próprio arquivo HTM; e c) com arquivo .css**



Fonte: print de tela do Navegador Google Chrome v88.0.

Note que o resultado entre integrar uma folha de estilos no arquivo HTML e externamente, por meio do uso da tag `link`, não resulta em uma página diferente. Isso ocorre porque o CSS utilizado permanece o mesmo, usa três seletores, todos utilizam como base a tag do HTML: `body`, `h1` e `p`.

Há, ainda, a possibilidade de aplicarmos um estilo *inline*, ou seja, na própria tag por meio do atributo: `<h1 style="color:blue;text-align:center;">`. Este é um *cabeçalho*`</h1>`, entretanto, este uso acaba restringindo o uso

do estilo, além de necessitar de mais esforços para a manutenção dos documentos HTML, visto que os estilos estão todos espalhados na *tag*. Ao se centralizar os estilos em arquivos .css, a manutenção se torna mais fácil.

Assim, com base nesses exemplos, podemos definir mais precisamente o que são as folhas de estilo em cascata, ou simplesmente, o CSS:

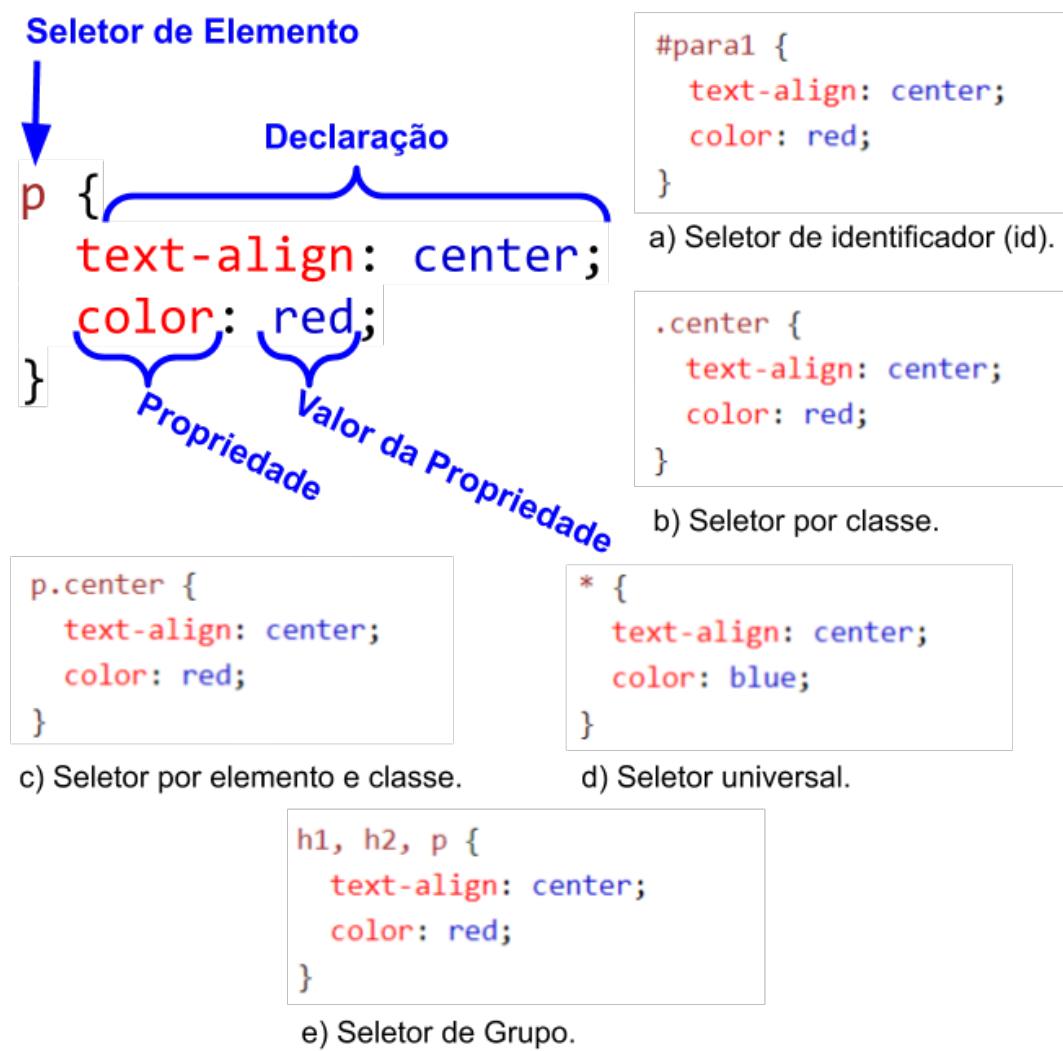
- CSS é usado para definir estilos de páginas web, incluindo seu design, layout e suas variações para diferentes tipos de dispositivos e tamanhos de tela. Em outras palavras, o CSS descreve como elementos HTML devem ser apresentados na tela, impressos, ou em outras mídias que exibam tais tipos de arquivos.
- CSS reduz o tempo de criação de páginas, uma vez que um único arquivo .css pode ser utilizado em múltiplas páginas (veremos com mais detalhes adiante).
- CSS não é uma linguagem de programação, tão pouco uma linguagem de marcação, como HTML. É uma linguagem de folhas de estilos, o que indica que sua função é aplicar estilos de modo seletivo a elementos nos documentos HTML.

A versão do CSS que utilizaremos é a 3, que implementa melhorias em relação a versão anterior, principalmente, no que diz respeito ao uso de imagens de plano de fundo, bordas arredondadas, transições e efeitos para criar animações de vários tipos (integradas e aperfeiçoadas com HTML5), antes implementadas com muitos esforços e utilizando-se de linguagens e bibliotecas como o JavaScript, que ainda é utilizado em vários outros contextos. Poderemos entender um pouco mais sobre como utilizar as regras do CSS para criação de nossas páginas.

## 1.1 Estrutura das regras das folhas de estilo em cascata.

Uma regra implementada, via CSS, é composta por diversos elementos, sendo os principais apresentados na Figura 2.

**Figura 2 – Regras de folhas de estilos e principais tipos de seletores**



Fonte: print de tela do Navegador Google Chrome v88.0.

Vejamos cada um deles:

- **Seletor:** indica qual elemento HTML, ou por meio de qual atributo (classe, id) uma propriedade será aplicada. No exemplo principal da Figura 2, o seletor é o próprio elemento *p* (tag HTML), que será modificado em relação a duas propriedades: *text-align*

(alinhamento de texto) e *color* (cor da fonte). Logo, todo parágrafo da página HTML terá seu estilo alterado para o indicado na definição do corpo da regra relacionado a tal seletor.

As mesmas propriedades poderiam ser aplicadas para uma tag parágrafo, utilizando o seletor de identificador (Figura 2, a): `<p id="para1">Texto do parágrafo.</p>`. Nesse caso, somente as tags *p* com identificador *para1* receberá tal estilo. Como um *id* é um identificador, espera-se que seja usado uma única vez no documento HTML.

Pode ser utilizado, ainda, o seletor de classe (Figura 2, b): `<p class="center">Texto do parágrafo.</p>`. Esse seletor permite que vários elementos recebam tal estilo, bastando apenas indicar o atributo *class* no elemento HTML desejado, referenciado no nome de tal seletor. No caso, o nome definido pelo desenvolvedor é *center*, mas é importante destacar que esse nome pode ser outro, de acordo com a preferência do desenvolvedor.

É importante destacar, ainda, que pode ser aplicada mais de uma classe para uma tag, ou seja, seria possível utilizar `class="center outra_classe"` e assim por diante, para aplicar mais de uma regra de estilo. Também é importante ressaltar que, nas folhas de estilo, enquanto o seletor de *id* é identificado pelo sustenido ("#"), a classe é identificada pelo ponto (".") e não deve ser nomeada iniciando com números.

Outro seletor que pode ser utilizado é a combinação da tag com a classe (Figura 2, c): `<p class="center">Texto do parágrafo.</p>`. Nesse caso, apenas elementos *p* com a classe *center* terão seus estilos modificados de acordo com as propriedades apresentadas na Figura 2.

Já o seletor “\*”, chamado de universal (Figura 2 d), aplicará o estilo para todos os elementos do documento HTML.

Finalmente, o seletor de grupo (Figura 2 e), como o nome diz, aplicará o estilo para todos os elementos especificados na relação de *tags* do seletor, separados por vírgula. Aqui, é importante mencionar que não se deve usar vírgula para separar os nomes das classes no atributo *class*, inserido no elemento do documento HTML. Isso significa que não se deve confundir o uso da vírgula do seletor de grupo com o uso de mais de uma classe no atributo *class* de um elemento.

- **Declaração:** delimita o *corpo* da regra ou as regras a serem aplicadas, considerando o seletor selecionado. Em outras palavras, cria um bloco que poderá receber uma ou mais propriedades que modificarão o estilo no documento HTML, de acordo com o seletor indicado antes da definição da declaração.
- **Propriedade:** é o meio pelo qual se estiliza um elemento HTML. No exemplo, temos o uso de duas propriedades, como já mencionado: *text-align* e *color*. Existem outras propriedades classificadas em grupos de propriedades aplicadas a determinados eventos. A regra definida por meio do CSS indicará quais propriedades serão afetadas no documento HTML, modificando sua aparência ou comportamento. Veremos as propriedades mais importantes, considerando alguns destes grupos de classificação, mais adiante.
- **Valor da propriedade:** é o valor que será dado à propriedade e que refletirá na modificação do (s) elemento (s) em relação aos seus estilos. No exemplo da Figura 2, a propriedade *color* recebe a cor *red* (vermelho). Logo, os elementos que receberem tal propriedade, com base no seletor, serão apresentados no website, na cor vermelha. Há diversos outros valores que podem ser utilizados em uma página. Esses valores são definidos na documentação oficial do CSS, na página da W3C (W3Schools, 2020).

Assim, devemos recapitular, em relação à uma regra de CSS, que:

- As linhas de comando devem ser envolvidas por chaves ("{}").
- Em cada declaração, deve-se utilizar dois pontos ":" para separar o nome da propriedade de seu valor.
- Dentro de cada conjunto de regras, deve-se utilizar o ponto e vírgula ";" para separar cada uma das propriedades.
- É indicado que cada propriedade seja apresentada em uma linha, facilitando a leitura e busca de alguma propriedade nos arquivos CSS.

Quantos arquivos CSS devem integrar a um website ou a aplicação web? Não há uma quantidade indicada de arquivos CSS. Entretanto, como boa prática, recomenda-se utilizar um arquivo CSS principal, com todas as propriedades mais genéricas e universais ao website, considerando todas as suas páginas e um CSS específico para cada página do projeto. Isso facilita os esforços de manutenção e testes.

A definição e criação de um conjunto de arquivos CSS para ser aplicado a um website, ou aplicação web, também auxilia a identificação de conflitos entre a aplicação de propriedades. Um conflito ocorre quando uma mesma propriedade é aplicada por seletores e conjuntos de regras diferentes a um mesmo elemento. Como há duas regras que alteram o estilo de um mesmo elemento, fica difícil identificar a origem de tal propriedade. Ao se trabalhar com mais de um arquivo, as identificações desses conflitos tornam-se mais fáceis.

Um conflito seria a aplicação de uma mesma propriedade, utilizando seletores diferentes. Por exemplo, considerando os exemplos apresentados na Figura 2. Se tivéssemos as linhas a seguir no documento HTML:

```
<p class="center">Texto do parágrafo.</p>
```

```
<p id="pcor">Texto do parágrafo.</p>
```

```
<p>Texto do parágrafo.</p>
```

E as seguintes regras:

```
p {
```

```
    color: red;
```

```
}
```

```
#pcor {
```

```
    color: blue;
```

```
}
```

```
p.center {
```

```
    color: green;
```

```
}
```

Não teríamos certeza de qual propriedade seria aplicada à cada um dos parágrafos. Nesse exemplo, a linha `<p class="center">Texto do parágrafo.</p>` receberia a propriedade do seletor “`p.center`” e teria um conflito com o seletor “`p`”. A linha `<p id="pcor">Texto do parágrafo.</p>` receberia a propriedade do seletor “`pcor`” e teria um conflito com o seletor “`p`”. E a linha `<p>Texto do parágrafo.</p>` não apresentaria nenhum conflito, já que não possui atributos de identificador ou de classe. Esses conflitos são facilmente visualizados, utilizando a ferramenta de desenvolvedor do navegador Google Chrome, por exemplo.

Ainda nesse contexto, é importante destacar que o CSS segue uma hierarquia para aplicação de estilos, segundo Laura (2019):

1. O estilo padrão do navegador.
2. O estilo definido pelos desenvolvedores.
3. Folha de estilos externa ("\*.css").
4. Folhas de estilos específicas na página, por meio da *tag style*, no cabeçalho do documento HTML.
5. Estilos *inline*, ou seja, os que são aplicados diretamente na *tag*.

Exemplo: `<p style="color:red">Texto do parágrafo.</p>`. Essa utilização, no entanto, é pouco indicada, pois ficará restrita à *tag* a qual foi aplicada e resultará em maiores esforços para identificar em que local o estilo foi alterado/ indicado.

Veremos, a seguir, algumas das propriedades mais utilizadas no CSS, para alteração dos estilos dos websites e aplicações web.

## 1.2 Principais propriedades em CSS

Primeiramente, para facilitar a documentação e criação de arquivos do tipo .CSS e sua posterior manutenção, devemos aprender a sintaxe dos comentários em CSS.

Um comentário em CSS é inserido como abaixo:

**`/* Bloco de comentário em CSS */`**

Esse comentário serve tanto para uma única linha, como no exemplo, quanto para um bloco com várias linhas. Seu uso é fortemente indicado, pois é por meio deles que pode ser adicionada alguma indicação ou nota ao código, bem como uma explicação quanto à finalidade da regra ou propriedades escritas.

O Quadro 1 apresenta as principais propriedades para formatação de textos. Algumas delas, como *background-color*, podem ser utilizadas em outros elementos HTML.

### Quadro 1 – Propriedades para formatação de textos

Propriedade	Descrição	Exemplo
<i>color</i>	Usada para configurar a cor de um texto. Aceita valores como nome da cor em inglês (“red”), valores hexadecimais (“#ff0000”) e valores da escala <i>Red-Green-Blue</i> (RGB) (“rgb(255,0,0)”).	body { color: blue; }  h1 { color: green; }
<i>background-color</i>	Altera a cor de fundo de um texto ou de um plano de fundo. Pode receber os mesmos tipos de valores da propriedade <i>color</i> .	body { background-color: lightgrey; color: blue; }  h1 { background-color: black; color: white; }

<i>text-align</i>	Utilizado para definir o alinhamento horizontal de um texto. Dentre os valores principais, pode receber “center” (centro), “left” (esquerda), “right” (direita) e “justify” (justificado).	h1 { text-align: center; }
<i>vertical-align</i>	Utilizado para definir o alinhamento vertical de um elemento (não apenas um texto). Pode receber os valores “top” (topo), “middle” (meio) e “bottom” (rodapé).	img.top {  vertical-align: top; }
<i>text-decoration</i>	Insere ou remove decorações de textos: “none” remove sublinhado de linha ou outros tipos de formatação; “overline” adiciona linha acima; “underline” adiciona linha abaixo (sublinhado).	a {  text-decoration: none; }  h1 {  text-decoration: overline; }

Fonte: adaptado e traduzido de W3Schools (2020).

O Quadro 2 apresenta propriedades de formatação de páginas em geral, com foco no plano de fundo das páginas HTML.

## Quadro 2 – Propriedades para formatação de páginas em geral

Propriedade	Descrição	Exemplo
<i>background-image</i>	Usado para definir uma imagem de fundo, permitindo a indicação de mais de uma imagem.	#example1 {  background-image: url(img_flwr.gif), url(paper.gif);  background-position: right bottom, left top;  background-repeat: no-repeat, repeat; }
<i>background-position</i>	Define a posição do plano de fundo.	
<i>background-repeat</i>	Define se a imagem do plano de fundo deve ( <i>repeat</i> ) ou não ( <i>no-repeat</i> ) se repetir.	
<i>background</i>	A propriedade <i>background</i> , assim como outras propriedades no CSS, podem receber vários valores que estilizarão o elemento. Ganha-se com a redução de propriedades, porém, é necessário entender quais valores estão inseridos, para fornecer manutenção posterior.	#example1 {  background: url(img_flwr.gif) right bottom no-repeat, url(paper.gif) left top repeat; }

Fonte: adaptado e traduzido de W3Schools (2020).

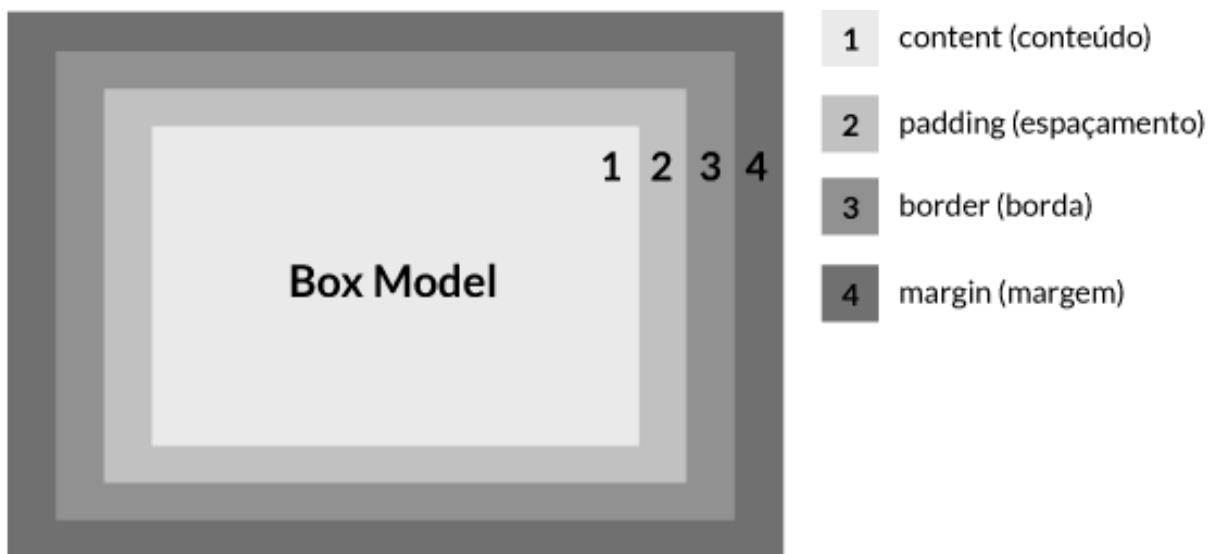
### Quadro 3 – *Box Model* ou Modelo de Caixa do CSS

Propriedade	Descrição	Exemplo
<i>border</i>	Usado para definir o estilo, a largura ( <i>width</i> ) e a cor das bordas de um elemento, como uma <i>div</i> , por exemplo. Geralmente, deriva dois elementos: <i>border-style</i> , <i>border-width</i> e <i>border-color</i> , como utilizado no exemplo ao lado.	div { height: 200px; width: 320px; padding: 10px; border: 5px solid gray; margin: 0; }
<i>margin</i>	Usado para definir um espaço ao redor de um elemento, fora de qualquer definição de borda. Pode ser aplicado somente para um lado específico da caixa, utilizando <i>margin-top</i> , <i>margin-bottom</i> , <i>margin-right</i> e <i>margin-left</i> . No exemplo ao lado, não há indicação de margem.	
<i>padding</i>	Usado para definir um espaço ao redor de um elemento, dentro de qualquer definição de borda. Ou seja, é interna à borda de uma caixa e pode ser aplicado para um lado específico da caixa: <i>padding-top</i> , <i>padding-right</i> , <i>padding-bottom</i> , <i>padding-left</i> .	
<i>height</i>	Usado para definir o comprimento (altura) de um elemento, seja uma caixa, figura, entrada de dados etc.	
<i>width</i>	Usado para definir a largura de um elemento, seja uma caixa, figura, entrada de dados etc.	

Fonte: adaptado e traduzido de W3Schools (2020).

A Figura 3 apresenta graficamente os elementos do modelo de caixas, especificado no Quadro 3 (BOX, 2020).

**Figura 3 – Modelo de Caixa do CSS**



Fonte: Box (2020).

A especificação do modelo de caixas, apesar de remeter ao uso de tabelas, quadros e outros elementos, é amplamente utilizado, pois este modelo coordena os demais elementos e suas estilizações para formar um website atraente.

Assim, as propriedades e as regras criadas dentro das folhas de estilo em cascata permitem alterar e estilizar os elementos, criando interfaces atrativas. Para isso, é importante conhecer não só as propriedades mencionadas aqui, mas também as demais especificadas pela W3C e disponibilizadas em seus portais (W3Schools, 2020; EIS; FERREIRA, 2012).

Bons estudos!



## Referências

BOX MODEL. **Tableless.** 2020. Disponível em: <https://tableless.github.io/iniciantes/manual/css/box-model.html>. Acesso em: 5 mar. 2021.

EIS, D.; FERREIRA, E. **HTML5 e CSS3 com farinha e pimenta.** São Paulo: Tableless, 2012.

LAURA, A. G. **XHTML/CSS:** criação de páginas web. São Paulo: Senac, 2019.

W3SCHOOLS. **HTML Reference.** 2020. Disponível em: <https://www.w3schools.com/tags/default.asp>. Acesso em: 5 mar. 2021.

# Linguagem JavaScript: do básico ao avançado.

Autoria: Anderson da Silva Marcolino

Leitura crítica: Gabriela Silveira



## Objetivos

- Conhecer os conceitos fundamentais, tipos de dados e variáveis, operadores e expressões, controle de fluxo com condicionais e loops.
- Compreender e utilizar funções, bem como manipular o documento *Object Model*.
- Conhecer as principais bibliotecas e *frameworks* JavaScript.

## 1. Introdução

A criação de websites com HTML5 e CSS3 permitem a criação de conteúdos que não são totalmente estáticos. Contudo, mesmo nessas novas versões, os comportamentos de elementos HTML e das folhas de estilo podem ser aprimorados, trazendo maior interação e comportamentos mais atrativos para os usuários finais de um website ou aplicação web. Entretanto, para isso, é necessário integrar um terceiro elemento: a linguagem de programação JavaScript.

No início dos anos 1990, a Netscape criou o Livescript, uma linguagem que buscava permitir a execução de scripts integrados a uma página web no próprio navegador. Como na época a linguagem Java despontava entre as linguagens de programação, Livescript foi rebatizada, recebendo o nome de JavaScript, com as devidas permissões da SUN, empresa mantenedora de Java na época e que hoje pertence à Oracle, visando impulsionar o uso de ambas as linguagens, segundo Silva (2020).

O nome do criador da linguagem JavaScript é Brendan Eich e a linguagem se tornou um padrão ECMA, em 1997. O que poucos sabem é que o nome oficial da linguagem é ECMAScript, mas que, devido a popularização do termo cunhado inicialmente, passou a ser chamada de JavaScript (W3SCHOOLS, 2020).

JavaScript é uma linguagem de programação interpretada, ou seja, o código não é compilador e gera um executável. Há a necessidade de um programa chamado interpretador, que lê o código e depois de sua interpretação (linha a linha e com análise léxica, sintática e semântica) a executa por demanda.

No contexto de JavaScript, o interpretador está integrado aos navegadores. Assim, a linguagem é suportada por todos os navegadores e é responsável pelo comportamento dinâmico das páginas. Sua potencialidade permite criar aplicações e websites impressionantes,

como as aplicações do Google. Contudo, como a linguagem possui grande tolerância a erros, a interpretação e resultado desta geram comportamentos que não eram esperados, resultando em muitos problemas.

Ademais, como o próprio nome sugere, JavaScript é uma linguagem de *scripting*. Logo, permite que o programador controle aplicações de terceiros. JavaScript, em especial, controla comportamentos dos navegadores por meio de elementos das páginas HTML e também aplicação e modificação de estilos (W3Schools, 2020).

## 1.1 Sintaxe do JavaScript

Primeiramente, para se integrar um script de JavaScript em um documento HTML pode-se utilizar diferentes métodos:

- No cabeçalho `<head>` ou no corpo `<body>` do documento:

```
<script>

    function myFunction() {

        document.getElementById("demo").innerHTML =
        "Parágrafo alterado./";

    }

</script>
```

- Em arquivo externo com extensão `.js`. Por exemplo: `meuscript.js`, integrado ao HTML:

No arquivo `meuscript.js`:

```
function myFunction() {
```

```
document.getElementById("demo").innerHTML =  
"Parágrafo alterado."  
}
```

No documento HTML (no corpo ou cabeçalho):

```
<script src="meuscript.js"></script>
```

JavaScript é uma linguagem *case sensitive*, ou seja, “**ultimoNome**” e “**ultimonome**” são variáveis diferentes. Uma linguagem considerada *case sensitive* é aquela que diferencia letras maiúsculas e minúsculas. Além disso, não é possível utilizar o traço (“-”) para separar e declarar duas variáveis. Pode-se utilizar o sublinhado (“ultimo\_nome”). Por convenção, os programadores adotam o padrão *Lower Camel Case*, ou seja, a primeira letra minúscula e as demais, das outras palavras em maiúsculo: “**ultimoNome**”, “**numeroDeAcesso**”.

A sintaxe do JavaScript define dois tipos de valores: os fixos, chamados de literais, e os valores que alteram, chamados de variáveis.

Os dois literais mais importantes são: os números, que podem ser escritos com ou sem casas decimais: 10.5 ou 1001, por exemplo; e o conjunto de caracteres (*strings*), que podem ser escritos com aspas duplas ou simples: “*Sala de aula*” ou ‘*Sala de aula*’.

As variáveis, em JavaScript, são utilizadas para armazenar valores. Para a declaração de uma variável, é necessário utilizar a palavra reservada “**var**”. Para atribuir um valor para uma determinada variável, utiliza-se o sinal de igual (“=”). No exemplo a seguir, uma variável *v* é definida e, então, o valor 8 é atribuído a mesma:

```
var v;
```

```
v = 8;
```

Antes de 2015, apenas a palavra “**var**” era utilizada para declarar variáveis. Em versões mais atuais surgem as palavras reservadas “**let**” e “**const**”. A palavra “**const**” é utilizada para declarar variáveis que não terão seus valores alterados, logo, podemos chamar de constante ao invés de variável, já que seu valor não irá variar. Por exemplo, uma variável do tipo “**const**” será declarada e receberá o valor que não será alterado: “**const x = 5;**”. Já “**let**”, é utilizado para declarar variáveis que são utilizadas em escopos restritos. Por exemplo, uma variável declarada com “**let**”: “**let x;**” em uma função, só estará disponível para uso dentro do corpo da função.

Existem ainda regras específicas para nomear uma variável ou constante:

- Nomes podem conter letras, dígitos, sublinhados e sinal monetário (“\$”).
- Nomes devem iniciar com letras, com “\$” e “\_”.
- Nomes variam ao usar letras maiúsculas ou minúsculas (*case sensitive*).
- Palavras reservadas, como “**let**”, “**var**” e outras não podem ser utilizadas como nomes de variáveis.

As variáveis em JavaScript são fracamente tipadas, o que significa que seu tipo dependerá do tipo de conteúdo que receberá: se numérico ou textual, salvo uma exceção. Vamos entender os três casos:

```
var x = "5"; //recebe valor textual
```

```
var x = 5; //recebe valor numérico ; note que aqui não temos as aspas que diferenciam texto de números
```

E a exceção do exemplo a seguir:

```
var x = "5" + 2 + 3;
```

Neste caso, a variável “x” se tornará uma variável textual. Os valores 2 e 3 serão concatenados junto ao texto 5, resultando em “523”.

É importante mencionar que existe ainda um tipo especial de variável, além das variáveis de cadeia de caracteres (*strings*), valor lógico (*boolean*) ou indefinido (*undefined*); que é o tipo objeto (*object*), utilizado para armazenar objetos:

```
var x = {primeiroNome:"Uzumaki", ultimoNome:"Naruto"};
```

No contexto das atribuições e cálculos, utilizando valores das variáveis, podemos utilizar vários operadores, como apresentado no Quadro 1.

**Quadro 1 – Operadores em JavaScript**

<b>Operador(es)</b>	<b>Descrição</b>	<b>Exemplo</b>
Aritméticos: +, -, *, **, /, %, ++, —	Adição, subtração, multiplicação, exponenciação, divisão, módulo (resto), incremento e decremento respectivamente.	<code>var z = x * y;</code>
Atribuição: =, +=, -=, *=, /=, %=, **=.	Atribuição, atribuição com soma, atribuição com subtração, atribuição com multiplicação, com divisão, com módulo e com exponenciação, respectivamente.	x += y é igual a:  <code>x = x + y;</code>

Operador de Cadeia de Caracteres ( <i>strings</i> ): +	O operador “+” pode ser utilizado para concatenar <i>strings</i> , assim como pode se utilizar o “+=”.	<code>var txt1 = "John"; var txt2 = "Doe";  var txt3 = txt1 + " " + txt2;  var txt1 = "Tenha um ";  txt1 += "ótimo dia!";</code>
Operadores de Comparação: ==, ===, !=, !==, >, <, >=, <=, ?.	Igual a, igual a (verifica se valor e tipo são iguais), não é igual, não é igual (valor e tipo), maior que, menor que, maior ou igual que, menor ou igual que e operador ternário, respectivamente.	X = 5;  X === 5 retorna verdadeiro ( <i>true</i> )  X === "5" retorna falso ( <i>false</i> )  "2" < "12" retorna falso ( <i>false</i> )  2 < "12" retorna verdadeiro ( <i>true</i> )
Operadores Lógicos: &&,   , !.	E lógico ou lógico e negação lógica, respectivamente.	<code>var x = 6, y = 3;  (x &lt; 10 &amp;&amp; y &gt; 1) retorna verdadeiro (<i>true</i>)</code>

Operadores de tipos: <i>typeof</i> e <i>instanceof</i>	Retorna o tipo de uma variável e retorna verdadeiro se um objeto é uma instância de um tipo de objeto, respectivamente. É importante destacar que o <i>typeof</i> retorna um <i>object</i> para vetores ( <i>arrays</i> ), pois um vetor é um objeto.	<code>typeof "John" // Retorna "string"</code>  <code>typeof 3.14 // Retorna "number"</code>  <code>typeof true // Retorna "boolean"</code>  <code>typeof false // Retorna "boolean"</code>  <code>typeof x // Retorna "undefined" (se x não tem nenhum valor)</code>  <code>typeof {name:'John', age:34} // Retorna "object"</code>  <code>typeof [1,2,3,4] // Retorna "object"</code>
---	---	---

Fonte: adaptado e traduzido de W3Schools (2020).

Agora que conhecemos os operadores mais utilizados, podemos conhecer os blocos de código mais utilizados em JavaScript: as funções.

Uma função é um bloco de código designado a executar uma tarefa particular. Uma função, em JavaScript, é executada quando algo a invoca (ou a chama). A Figura 1 apresenta um exemplo de função que efetua um cálculo e retorna um resultado.

## Figura 1 – Exemplo de função com JavaScript

```
<!DOCTYPE html>
<html>
<body>

<h2>Função em JavaScript</h2>

<p>Este exemplo chama uma função que executa um cálculo, retornando o resultado:</p>

<p id="demo"></p>

<script>
function minhaFuncao(p1, p2) {
    return p1 * p2;
}
document.getElementById("demo").innerHTML =
minhaFuncao(4, 3);
</script>

</body>
</html>
```

### Função em JavaScript

Este exemplo chama uma função que executa um cálculo, retornando o resultado:

12

Fonte: elaborada pelo autor.

A Figura 1 apresenta, primeiramente, uma página em HTML5, sem o cabeçalho, já que este é opcional, com um script JavaScript no corpo da página. Esse script é iniciado a partir do momento que é interpretado pelo navegador, altera o documento de modelo de objetos (DOM) do HTML, inserindo o resultado calculado na função “***minhaFuncao()***” no elemento de parágrafo com identificador “***demo***”, de demonstração.

Analizando especificamente o que temos no elemento script, podemos notar, primeiramente, a declaração da função, nas linhas:

```
function minhaFuncao(p1, p2) {

    return p1 * p2;

}
```

Note que a função recebe dois parâmetros: p1 e p2, utilizados na multiplicação que, após calculada, resultará no retorno do valor calculado. Para que o retorno seja exibido na página HTML o JavaScript

utiliza um meio de alterar o DOM do HTML, exibindo tal valor. Isso ocorre por meio da linha:

```
document.getElementById("demo").innerHTML =  
minhaFuncao(4, 3);
```

Primeiro, obtém-se o elemento HTML, que será inserido o resultado por meio da referência ao documento HTML (“*document*”) seguido por uma função que busca de um elemento do HTML pelo seu *id* (“*getElementById*”) e seguindo pelo “*innerHTML*”, que indica que o valor deverá ser exibido dentro do elemento HTML que tenha ***id = “demo”***, correspondendo, no caso, pelo elemento de parágrafo. Na sequência, há a chamada da função com a passagem dos parâmetros numéricos 4 e 3: “***minhaFuncao(4, 3);***”.

JavaScript utiliza-se de alguns meios diferentes para mostrar as saídas, ou seja, os retornos de suas funções e conteúdos executados, sendo:

- ***innerHTML***: utilizado para acessar definir um conteúdo de um determinado elemento do HTML. Para isso, deve ser utilizado juntamente com o método ***document.getElementById(id)***, onde ***id*** corresponde ao atributo de identificação do elemento desejado para ter seu conteúdo alterado.
- ***document.write()***: usado com propósito de testes, exibirá a saída na página. Deve ser incluído sempre dentro do elemento de corpo ou cabeçalho, pois, se for colocado após um documento HTML ser carregado, sobrescreverá e exibirá somente a saída do JavaScript, ignorando o conteúdo em HTML. Exemplo: `<button type="button" onclick="document.write(5 + 6)">Testar!</button>`.
- ***window.alert()***: exibe uma janela de alerta. Pode ser utilizado diretamente, sem a palavra ***window***, já que seus métodos e variáveis são pertencentes ao objeto ***window***. Exemplo: `<script>`

```
window.alert(5 + 6);</script> ou <script> alert(5 + 6);</script>.
```

- **console.log()**: utilizado para depuração, exibe mensagens e resultados no console do navegador. Exemplo: <script>console.log(5 + 6);</script>.

Assim, podemos retornar à sintaxe das funções, que possui como base, para a sua definição, o seguinte modelo:

```
function nome(parametro1, parametro2, parametro3) {  
    // código a ser executado  
}
```

A palavra reservada para declarar uma função é **function**, seguida do nome da função e parênteses. As regras para definirem um nome de uma função são as mesmas das variáveis. Finalmente, dentro dos parênteses podem ser incluídos um ou mais parâmetros ou argumentos, que se comportam como variáveis locais, ou seja, visíveis apenas dentro do corpo da função, que é definido pelas chaves “{}”. As funções são equivalentes a sub-rotinas e procedimentos em outras linguagens de programação. É importante mencionar que qualquer variável declarada com **var** ou **let** dentro da função, será visível apenas para o código dentro de tal função.

O código do corpo da função será executado quando um algo chamar ou invocar a respectiva função, como por meio do nome da função no atributo **onclick** de um botão; quando chamado diretamente de outro código JavaScript, é comum uma função chamar outra função; e também de modo automático, como visto anteriormente na Figura 1. As chamadas de funções e códigos JavaScript são chamados de eventos,

que são atributos disponíveis nos elementos HTML e que permitem tal chamada. O Quadro 2 apresenta os eventos HTML mais comuns.

### Quadro 2 – Eventos HTML

Evento	Descrição
<i>onchange</i>	Um elemento HTML foi modificado.
<i>onclick</i>	O usuário clicou em um elemento HTML.
<i>onmouseover</i>	O usuário moveu o mouse sobre um elemento HTML.
<i>onmouseout</i>	O usuário moveu o mouse para fora de um elemento HTML.
<i>onkeydown</i>	O usuário pressionou alguma tecla do teclado.
<i>onload</i>	O navegador terminou de carregar uma página.

Fonte: adaptado e traduzido de W3Schools (2020).

Considerando a lista de eventos HTML, podemos listar uma série de vantagens que podem ser obtidas pelo uso de tais eventos com JavaScript:

- Funções ou outras execuções que precisam ser feitas toda vez que a página é recarregada ou fechada.
- Ações que devem ser executadas quando um usuário clica em um botão ou elemento HTML.
- Validação de conteúdos que são inseridos em campos de entradas de dados.

Um outro elemento que, quando encontrado no corpo da função, gera um comportamento diferente, é uma declaração de ***return*** (retorno). Ao encontrar um ***return***, a função terá sua execução finalizada. Se a função foi chamada a partir de uma instrução, o JavaScript retornará para executar o código após a instrução de chamada. As funções, geralmente, calculam um valor de retorno. O valor de retorno é “retornado” de

volta ao “*chamador*”, como é o caso do exemplo a seguir, que calcula o produto de dois números:

```
var x = minhaFuncao(4, 3); //Aqui a função é chamada, e será  
finalizada ao retornar um valor  
  
function myFunction(a, b) {  
  
    return a * b; //Função retorno o produto de a e b  
  
}
```

Note que estamos invocando a função e armazenando no valor na variável **x**. Poderíamos querer passar a declaração da função, seu objeto, para uma outra variável. Para isso, devemos tirar os parênteses “()”, pois garantem que seja chamada a função, retornando o resultado de sua execução.

Como considerações finais sobre as funções, podemos indicar as vantagens de seu uso: reuso, já que podemos definir uma função uma vez, e reusá-la quantas vezes forem necessárias; e utilizar um mesmo código, com argumentos diferentes, para produzir resultados diferentes, o que também está incluído no contexto maior de reutilização.

Por fim, um outro elemento importante, onde também utilizamos as funções, é o conceito de objetos. Como ocorre na orientação à

objetos, seu uso permite a criação de objetos da vida real, por meio da designação de suas propriedades e de seus métodos, ou seja, de suas características e suas ações, como pode ser observado no Quadro 3.

**Quadro 3 – Objetos em JavaScript**

Objeto	Propriedades	Métodos
 Fonte: <a href="https://www.w3schools.com/js/objectExplained.gif">https://www.w3schools.com/js/objectExplained.gif</a>	carro.nome = Fiat carro.modelo = 500 carro.peso = 850kg carro.cor = branco	carro.darPartida() carro.dirigir() carro.freiar() carro.parar()
<b>Objeto declarado em JavaScript</b>		
<pre>var carro = {   nome: "Fiat",   modelo: "500",   peso: "850kg",   cor: "branco",   darPartida: function(){ //corpo da função},   freiar: function(){ //corpo da função},   parar: function(){ //corpo da função} }</pre>		

Fonte: adaptado e traduzido de W3Schools (2020).

Considerando o exemplo de objeto do Quadro 3, todos os carros possuem as mesmas propriedades, mas os valores das propriedades podem se diferenciar de um carro para outro. O mesmo vale para os métodos: todos os carros possuem os mesmos métodos, contudo estes são executados em momentos diferentes. As propriedades e métodos são declarados sempre como um par de nome e valor, separados por

dois pontos “：“, logo, podemos falar que um objeto em JavaScript são contêiners para nomes e valores chamados propriedades ou métodos.

Há dois modos de acessar um valor de um objeto: **nomeObjeto.propriedadeNome** ou **nomeObjeto["propriedadeNome"]**. Já para acessar um método podemos usar **nomeObjeto.nomeMetodo()**. E tal como funções, se removermos os parênteses, retornaremos à declaração do método.

Ainda no contexto de orientação a objetos, temos o uso da palavra reservada **this**. Em uma definição de função, **this** se refere ao dono de uma função. Veja o trecho de código a seguir:

```
var pessoa = {  
    primeiroNome: "John",  
    ultimoNome: "Doe",  
    id: 5566,  
    nomeCompleto: function() {  
        return this.primeiroNome + " " + this.ultimoNome;  
    }  
};
```

**This** indica que o objeto pessoa possui uma função chamada **nomeCompleto**. Em outras palavras, **this.primeiroNome** faz referência à propriedade **primeiroNome** deste objeto (em inglês “of this object”). Resumindo, é um meio de referenciar o próprio objeto no qual um atributo ou método precisa ser usado ou referenciando.

Tomando como exemplo as propriedades de cadeia de caracteres, devemos nos atentar quanto algumas restrições e características especiais das **strings**, principalmente às relacionadas ao uso de caracteres especiais. Por exemplo, para obter o resultado de um texto entre aspas duplas, é necessário colocar a barra invertida “\”:

```
var x = "Somos chamados de \"Vikings\" do Norte.;"
```

A mesma barra invertida deve ser utilizada para imprimir aspas simples (“\"") e a própria barra invertida (“\\”). Existe ainda outras sequências válidas em JavaScript, com destaque para **\b** para representar um espaço (*backspace*) e **\n** para uma nova linha.

Uma outra característica importante das cadeias de caracteres em JavaScript, é que estas podem ser criadas como uma instância de **String**:

```
var x = "Maria";
```

```
var y = new String("Maria");
```

```
var z = new String("Maria");
```

A palavra reservada **new** instancia um objeto do tipo **String** tanto para a variável **y**, quanto **z**. Apesar da criação de uma **string** por meio de tal recurso não é recomendada a utilização do mesmo, já que reduz o desempenho da execução e pode dificultar a comparação de valores de uma variável. Por exemplo, se compararmos, com base nas variáveis do trecho de código acima **x == y** retornará verdadeiro, pois possuem valores iguais. Se utilizarmos a comparação **x === y**, teremos falso, pois, apesar dos valores serem iguais, os tipos serão diferentes. Finalmente, se compararmos **y** com **x**, utilizando **==** ou **===**, em ambas as comparações, será retornado falso, visto que quando objetos são instanciados, ainda que o conteúdo seja igual, sua instância, como um todo, não o é.

Nesse contexto, você pode estar se perguntando o porque da existência dos objetos do tipo ***string***, e a resposta é simples: seus métodos.

Os métodos para trabalhar com ***strings*** em JavaScript considera valores primitivos de ***strings***, como na definição da variável **x**, que recebe o nome “**Maria**” anteriormente, como objetos. Assim, é possível utilizar as propriedades e métodos dos objetos de ***string*** também com tais variáveis.

Os principais métodos utilizados são:

- ***length***: utilizado para identificar o tamanho de uma ***string***.

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
```

```
var sln = txt.length;
```

- ***indexOf()***: utilizado para retornar o índice da primeira ocorrência de uma determinada palavra ou caractere em uma ***string***, lembrando que a contagem das posições começa sempre do zero.

```
var str = "localize onde 'localize' ocorre! ";
```

```
var pos = str.indexOf("localize");
```

- ***lastIndexOf()***: retorna o índice da última ocorrência de um texto em uma ***string***.

```
var str = "localize onde 'localize' ocorre! ";
```

```
var pos = str.lastIndexOf("localize");
```

Algumas observações sobre os métodos ***indexOf()*** e ***lastIndexOf()***: a) ambos retornam -1 se o texto indicado não estiver presente na ***string***; b) ambos os métodos aceitam um segundo parâmetro (`var pos = str.indexOf("localize", 15);`), que indica a partir de que posição a busca

deve iniciar. Por exemplo, considerando o método ***lastIndexOf()***, se o segundo parâmetro é 15, a busca iniciará na posição 15 e, a partir desta posição, buscará reduzindo as posições até o encontro do termo indicado.

Assim como as ***strings***, os números também possuem algumas observações. As principais são os resultados de uma operação não permitida, que, geralmente, envolvem tipos de dados diferentes. Por exemplo, o ***NaN*** (*Not a Number*), em português, não é um número, ocorre quando se tenta dividir um número por uma ***string***, por exemplo. O ***-Infinity***, apresentado quando o valor de uma operação resulta em um número que ultrapassa o limite de representação de uma variável em JavaScript (padrão IEEE 754), que é de um número de ponto flutuante de 64 bits, onde a fração vai de 0 a 51, o expoente de 52 a 62 e o sinal no bit 63.

Em relação aos métodos para se trabalhar com números, temos o ***toString()***, que converte um valor em cadeia de caracteres (`var x = 123; x.toString(); (123).toString();`); o ***toExponential()***, que calcula o exponencial de uma variável pelo número passado por parênteses (`var x = 9.656; x.toExponential(2); // retorna 9.66e+0`); e ***toFixed()***, que retorna um valor em ***string*** com a precisão das casas decimais passadas como parâmetros (`var x = 9.656; x.toFixed(0); // retorna 10 x.toFixed(2); // retorna 9.66`).

Assim, pudemos aprender a respeito dos conceitos básicos sobre JavaScript e as principais funcionalidades que podem ser integradas às páginas HTML e aplicações web. Existem outros objetos e métodos que facilitam nossa vida ao utilizarmos JavaScript. Desse modo, é recomendado utilizar sempre a documentação padrão de JavaScript e, principalmente, conteúdos divulgados pelo consórcio do *World Wide Web* que fornecem conteúdos atuais e completos sobre tudo que é necessário para se desenvolver utilizando JavaScript e tecnologias web! (W3Schools, 2020; SILVA, 2022). Bons estudos!



## Referências

SILVA, M. S. **JavaScript-Guia do Programador**: guia completo das funcionalidades de linguagem JavaScript. São Paulo: Novatec, 2020.

W3SCHOOLS. **HTML Reference**. 2020. Disponível em: <https://www.w3schools.com/tags/default.asp>. Acesso em: 5 mar. 2021.

# Implementação de Páginas Web com HTML, CSS e JavaScript.

Autoria: Anderson da Silva Marcolino

Leitura crítica: Gabriela Silveira Barbosa



## Objetivos

- Aplicar os conceitos fundamentais sobre HTML, CSS e JavaScript.
- Compreender conceitos avançados e como criar uma página responsiva.
- Criar validações para formulários HTML com JavaScript.



# 1. Introdução

A criação de websites de qualidade resulta na integração de diferentes tecnologias para web, sendo as principais a linguagem de marcação de hipertexto (HTML), as folhas de estilo em cascata (CSS) e o dinamismo trazido pela linguagem de programação ECMA Script, comumente chamada de JavaScript.

Para aplicar os conceitos básicos e compreender conceitos avançados, evoluindo também nossos projetos, devemos ter uma boa base de conhecimento dos elementos essenciais. Para isso, construiremos um projeto de uma página web com algumas funcionalidades diferentes:

1. Uma página principal que dá acesso a outras ferramentas.
2. Uma página com um formulário para cadastrar tarefas.

Essas páginas serão desenvolvidas utilizando HTML, CSS e JavaScript e, em um segundo momento, utilizaremos, em parte de nossa interface, o *framework Bootstrap*, para conhecermos sua utilização básica. O *Bootstrap* é um *framework* web de código aberto, utilizado para o desenvolvimento de componentes de interface e *front-end* para *websites* e aplicações web que utilizam HTML, CSS e JavaScript.

## 1.1 Estrutura do documento HTML

A Figura 1 apresenta a estrutura básica do nosso arquivo HTML, que será adaptado e modificado na medida em que nosso projeto avança.

## Figura 1 – Estrutura básica em HTML

```

1   <!DOCTYPE html>
2   <html lang="pt-BR">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <meta name="author" content="Nome do autor">
7       <meta name="description" content="Template inicial HTML puro.">
8       <title>Título da Barra de Títulos</title>
9   </head>
10  <body>
11
12  </body>
13  </html>

```

1

```

10  <body>
11      <header>
12          <h1>Cabeçalho do Site</h1>
13      </header>
14      <nav>
15          <ul>
16              <li><a href="#">Início</a></li>
17              <li><a href="#">Cadastro</a></li>
18              <li><a href="#">Tarefas</a></li>
19          </ul>
20      </nav>
21      <section id="secao1">
22          <header>
23              <h2>Subtítulo (Seção)</h2>
24          </header>
25          <article>
26              <p>Conteúdo da seção.</p>
27          </article>
28          <footer>
29              <p>Rodapé da Seção</p>
30          </footer>
31      </section>
32      <aside><p>Conteúdo da coluna lateral.</p></aside>
33      <footer>
34          <p>Rodapé da Página</p>
35      </footer>
36  </body>
37  </html>

```

2

### Cabeçalho do Site

- [Início](#)
- [Cadastro](#)
- [Tarefas](#)

### Subtítulo (Seção)

Conteúdo da seção.

Rodapé da Seção

Conteúdo da coluna lateral.

Rodapé da Página

3

Fonte: print de tela do Visual Studio Code.

A Figura 1, item 1, apresenta a estrutura básica do documento HTML. Nela, podemos identificar o cabeçalho identificado pela tag `<head>`.

Este elemento é opcional no HTML5, porém, se optarmos por suprimir tal, as *metatags* que fornecem dados sobre o site devem ser mantidas, no entanto, fora de um elemento de cabeçalho. Adicionalmente, as *metatags author* e *description* são duas das *metatags* utilizadas para auxiliar mecanismos de busca da página, como os da Google. Finalmente, ainda no mesmo item temos o corpo da página (*body*), onde o conteúdo é inserido e exibido no item 2.

A Figura 1, item 2, apresenta, como primeiro elemento do conteúdo da página, o cabeçalho (*<header>*), seção que é o topo da página e que receberá o título do site ou uma imagem de cabeçalho. Na sequência, temos o elemento de navegação, que resulta em menu de conteúdo, por meio de uma lista não ordenada, criada com os elementos *<ul>* e *<li>*, sendo a tag *<ul>* a que inicia a lista, e a tag *<li>* a que indica um item da lista. Na figura, temos três itens utilizados para compor o menu.

Ainda no contexto do corpo da página, temos uma seção e seus elementos internos. Uma seção refere-se, como o nome sugere, a um conteúdo que busca representar uma seção textual de um artigo. No exemplo, o artigo (*<article> ... </article>*) é um elemento interno da seção, mas pode ser trocado tranquilamente com a mesma. A seção é formada pelas tags *<section>* e *</section>* e, internamente possui o cabeçalho, o artigo em si, e o rodapé (*<footer> ... </footer>*). Note que estes são os mesmos elementos utilizados no corpo da página, e possuem a mesma função.

Esses elementos são, muitas vezes, substituídos pelas *divs*. O elemento de divisão HTML *<div>* é um container genérico para conteúdo de fluxo, que, de certa forma, não representa nada. Pode ser utilizado para agrupar elementos para fins de estilos (usando *class* ou *id*), ou porque compartilham valores de atributos (MDN, 2019).

Assim, para cada um dos elementos específicos, que criam as seções dos documentos HTML, uma *div* com uma classe ou identificador pode

ser utilizada no lugar. A formatação, ou seja, seu estilo, dependerá exclusivamente das regras das folhas de estilos aplicadas ao documento. Em alguns contextos, o uso de *divs* pode resultar em uma desvantagem: os elementos semânticos específicos do HTML podem receber um comportamento especial pelo navegador. Desse modo, como o elemento *div* é genérico, esses comportamentos não serão aplicados, pois só serão reproduzidos por meio das folhas de estilos e não diretamente pelo HTML.

Como resultado dos elementos no documento HTML, salvo no arquivo “*index.html*” com este nome, pois se trata da página principal do documento, temos o resultado apresentado na Figura 1, item 3. Uma página desenvolvida puramente com HTML5 que pode, então, ser modificada com folha de estilos em cascata para ficar melhor apresentável.

## 1.2 Aplicando folha de estilos em cascata

O primeiro passo, para criarmos nossas folhas de estilos de nossa página, é criar um arquivo com extensão “.css”. Por padrão, colocaremos todos os arquivos de folhas de estilos em uma subpasta chamada “css”, no diretório raiz de nosso projeto.

Podemos criar quantos arquivos css forem necessários, porém, utilizaremos apenas um para nosso projeto, o “*style.css*”. A seguir, é apresentado o excerto de código em HTML que importará nosso CSS.

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

O código a seguir apresenta as regras que serão referenciadas no documento css e integradas ao documento HTML e suas respectivas explicações.

```
@import url("https://fonts.googleapis.com/  
css?family=Roboto:400,500,700&display=swap");
```

O código anterior permite importar uma fonte para o site, que é, posteriormente, referenciada como valor da propriedade *font* na regra a seguir, do corpo da página.

```
body {  
  
    margin: 0;  
  
    padding: 0;  
  
    font: 14px "Roboto", sans-serif;  
  
    color: #333;  
  
    background: #eaebec;  
  
    -webkit-font-smoothing: antialiased !important;  
  
}
```

A regra do bloco definido pelo seletor *body* remove qualquer margem e espaço em branco existente, define o tamanho da fonte e a família da fonte a ser utilizada (propriedade *font*). É definido, ainda, uma cor para o texto (propriedade *color*) e uma cor de fundo para a página (propriedade *background*). Essas cores serão alteradas posteriormente, bem como as demais. Foram indicadas para facilitar a visualização do resultado das regras de nossas folhas de estilo. A última propriedade busca garantir o efeito de *Anti-Aliasing*, ou seja, remover possíveis serrilhados que possam ser apresentados na fonte, para navegadores que tenham como *engine*, o *webkit*.

```
.cabecalho {  
    height: 40px;  
    padding: 30px;  
    text-align: left;  
    background: red;  
    overflow: hidden;  
}
```

A regra do bloco, definido pelo seletor de classes “.cabecalho”, aplica a estilização para o cabeçalho da página, como pode ser observado na Figura 2, destacado em vermelho. A propriedade *overflow* trata o que ocorre com o conteúdo que ultrapassa determinada área. Além do valor da propriedade *hidden*, que esconde o conteúdo que excede o tamanho do elemento, temos os valores *visible*, que é o valor padrão, que exibe o conteúdo normalmente; o *scroll*, que adiciona barras de rolagem para que seja possível visualizar o conteúdo excedente; e o *auto*, que exibe as barras de rolagem somente quando o conteúdo exceder o elemento. Finalmente, é importante mencionar que essa propriedade só funciona em elementos que tenham a altura (*height*) definida.

Assim como o seletor do cabeçalho, o seletor para a classe “.menu” especifica regras de cor de fundo e de exibição do conteúdo. O seletor “.menu ul”, “.menu ul, li”, “.menu li, a” e “.menu a” alteram as propriedades dos elementos que são exibidos no menu, sendo, respectivamente, as características de lista do *.menu (ul)*, de itens de lista do *.menu(li)* e de links do *.menu (a)*. Em especial, removem a formatação da lista de elementos (*text-decoration: none*), exibem os itens horizontalmente (*display: inline-block*) e ajustam as margens e espaços ao redor dos elementos do menu.

```
.menu {  
background-color: green;  
overflow: hidden;  
}  
  
.menu ul{  
margin-right: 10%;  
margin-left: 10%;  
}  
  
.menu ul, li{  
padding: 0;  
margin: 0;  
}  
  
.menu li, a {  
color: #ffffff;  
text-decoration: none;  
display: inline-block;  
}  
  
.menu a {  
padding: 15px 30px;
```

}

O seletor “*.menu a:hover*” é um seletor especial. Os dois pontos e a palavra *hover* indicam que, ao posicionar o cursor do mouse sobre o link da nossa página (elemento *a*), uma estilização específica será aplicada. No caso, o fundo ficará da cor branca e o texto ficará da cor preta.

```
.menu a:hover{  
  
    background-color: white;  
  
    color: black;  
  
}
```

O seletor da classe “*.row*” define uma linha na qual serão posicionados os elementos identificados por meio do identificador “*secao1*” e “*secao2*”. Estas duas seções recebem a propriedade *float*, que garante que o elemento flutue para a esquerda, quando a resolução for modificada, tornando os elementos responsivos, ou seja, são deslocados para garantir a visualização do seu conteúdo, ainda que em dispositivos com resolução menor.

Essa responsividade também é garantida indicando que o tamanho (largura) do conteúdo seja um percentual que é calculado com base na resolução. A classe “*.row::after*” possui um seletor que insere algo depois (*after*) do conteúdo do elemento em questão, no caso, o com classe “*.row*”. A regra aplicada busca eliminar a formatação do *float*, que foi aplicada às seções anteriores (“*secao1*” e “*secao2*”).

```
.row {  
  
    padding: 10px;  
  
}
```

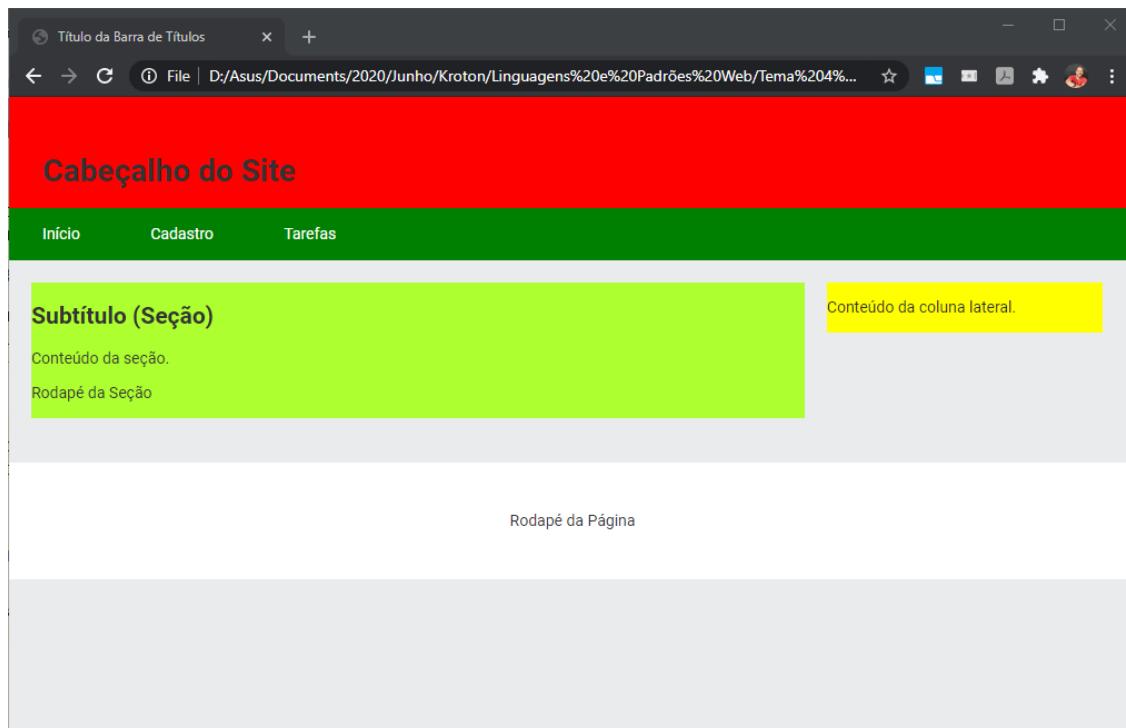
```
#secao1 {  
    float: left;  
  
    width: 70%;  
  
    background-color: greenyellow;  
  
    margin: 10px;  
  
}  
  
#secao2 {  
    float: left;  
  
    width: 25%;  
  
    background-color: yellow;  
  
    margin: 10px;  
  
}  
  
.row::after {  
    content: “”;  
  
    display: table;  
  
    clear: both;  
  
}
```

A última regra inserida em nosso arquivo “style.css” refere-se ao rodapé de nossa página, que, ao contrário das seções anteriores, não faz parte da *div* identificada pela classe “row”.

```
.rodape {  
    padding: 30px;  
  
    background: white;  
  
    text-align: center;  
  
    margin-top: 20px;  
  
}
```

Com base nas regras definidas pelo CSS, temos o resultado parcial de nossa página inicial, de acordo com a Figura 2.

**Figura 2 – Página HTML com CSS**



Fonte: print de tela do Navegador Google Chrome v88.0.

Retomando nossas tarefas, podemos concluir que possuímos uma página inicial que, claro, necessita receber conteúdos e ficar visualmente melhor, por meio da mudança de esquemas de cores. Isso, podemos modificar por conta própria. O que precisamos, agora, é criar as duas outras páginas, configurá-las para que estejam acessíveis nos links do menu, bem como a página principal. Para isso, criaremos o arquivo *pagina1.html* dentro de um subdiretório chamado “*html*” em nosso projeto, que deverá receber o mesmo conteúdo da página *index.html*, exibida na Figura 2.

Já o código do menu, ficará como segue:

Na *index.html* (armazenado na pasta raiz de nosso projeto):

```
<nav class="menu">  
  <ul>  
    <li><a href="index.html">Início</a></li>  
    <li><a href=".//html/pagina1.html">Tarefas</a></li>  
  </ul>  
</nav>
```

Na *pagina1.html* (armazenada na subpasta *html* de nosso projeto):

```
<li><a href=".//index.html">Início</a></li>  
<li><a href="pagina1.html">Cadastro</a></li>
```

Note que o caminho dos arquivos depende do diretório em que estão localizados. Os “..” (dois pontos) levam a um diretório anterior, o “.” (um ponto) acessa um subdiretório.

Na sequência, criaremos nosso formulário, na página 1, e nosso cadastro de tarefas, na página 2, destacando os códigos em JavaScript.

## 1.3 Integrando JavaScript

O cadastro de tarefas estará disponível na *pagina1.html*, logo, será necessário fazer algumas alterações no código HTML, principalmente por utilizarmos a folha de estilos do *framework Bootstrap 4*.

As linhas a seguir indicam as principais modificações do arquivo HTML (*pagina1.html*):

- Inclusão no elemento *head*:

```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
```

- Código inserido dentro da *div*, identificado pela classe “*row*”, o código existente anteriormente, vindo da página *index.html*, deverá ser substituído:

```
<div class="container">

  <div class="card">

    <h1 class="card-header">Lista de Tarefas</h1>

    <div class="card-body">

      <div class="input-group mb-3">

        <input type="text" class="form-control"
          placeholder="Digite a tarefa" name="tarefa"
          autofocus>
```

```
<div class="input-group-append">  
    <button class="btn btn-primary" type="button"  
        id="botao">Cadastrar</button>  
</div>  
  
</div>  
  
<ul id="lista" class="list-group">  
</ul>  
  
</div>  
  
</div>  
  
<script src="../js/script.js"></script>
```

Note que o código desenvolvido em JavaScript está incluso no subdiretório “js”, no arquivo “script.js”. O arquivo .js apresenta o código a seguir:

```
let input = document.querySelector('input[name=tarefa]');  
  
let btn = document.querySelector('#botao');  
  
let lista = document.querySelector('#lista');  
  
let card = document.querySelector('.card');  
  
let tarefas = JSON.parse(localStorage.getItem('tarefas')) || [];
```

O código anterior cria as variáveis que serão utilizadas no decorrer do código. Temos os *querySelector*, que buscam elementos pelo texto inserido entre parênteses e como parâmetros. A variável que se difere das demais é a “*tarefas*”, que cria um vetor com as tarefas registradas. Note que o conceito de seletores em JavaScript são o que criam a integração entre os elementos HTML e as funções programadas em tal linguagem.

Na sequência, temos a função *renderizarTarefas()*, que cria uma lista de itens que serão exibidos. Essa lista receberá todas as tarefas inseridas. Temos, no corpo desta função, um *for* que percorre todas as tarefas, povoando a lista e criando-a.

```
function renderizarTarefas(){

    lista.innerHTML = "";

    for(tarefa of tarefas){

        let itemLista = document.createElement('li');

        itemLista.setAttribute('class', 'list-group-item list-group-item-action');

        itemLista.onclick = function(){

            deletarTarefa(this);

        }

        let itemTexto = document.createTextNode(tarefa);

        itemLista.appendChild(itemTexto);

        lista.appendChild(itemLista);

    }

}
```

```
    }  
  
}  
  
renderizarTarefas();
```

A seguir, temos a variável *btn*, que referencia o botão e que, no momento em que este for “clicado” pelo usuário, ativará a função que, por meio de uma estrutura condicional *if*, verifica se a *novaTarefa* não está vazia e adiciona no *LocalStorage*, por meio da função *salvarDadosNoStorage()*.

A chamada da função *removerSpans()* cria um alerta com possíveis problemas, caso o texto da tarefa esteja vazio, exibindo ainda uma mensagem para que o usuário entre com tal texto.

```
btn.onclick = function(){  
  
    let novaTarefa = input.value;  
  
    if(novaTarefa !== ""){  
  
        tarefas.push(novaTarefa);  
  
        renderizarTarefas();  
  
        input.value = " ";  
  
        removerSpans();  
  
        salvarDadosNoStorage();  
  
    }else{  
  
        removerSpans();  
    }  
}
```

```
let span = document.createElement('span');
span.setAttribute('class', 'alert alert-warning');

let msg = document.createTextNode('Você precisa digitar a
tarefa que deseja registrar!');

span.appendChild(msg);

card.appendChild(span);

}

}

function removerSpans(){

let spans = document.querySelectorAll('span');

for(let i = 0; i < spans.length; i++){

card.removeChild(spans[i]);

}

}
```

A função *deletarTarefa()* é adicionada à tarefa que foi acrescentada, por meio da função *renderizarTarefas()*. Faz com que a tarefa da lista, ao ser “clicada” pelo usuário, seja excluída.

```
function deletarTarefa(tar){

tarefas.splice(tarefas.indexOf(tar.textContent), 1);

renderizarTarefas();
```

```
    salvarDadosNoStorage();  
}  
}
```

Finalmente, a última função coloca, no *localStorage*, a tarefa digitada pelo usuário. O *localStorage*, ou armazenamento na web, também conhecido como armazenamento DOM, fornece ao desenvolvedor a possibilidade de armazenar dados no lado do cliente. É como se tivéssemos um banco de dados local, inserido no próprio navegador, que mantém as informações armazenadas até serem totalmente removidas pelo usuário. Assim, já temos um projeto inicial que trabalha, inclusive, com o *localStorage*.

```
function salvarDadosNoStorage(){  
    localStorage.setItem('tarefas', JSON.stringify(tarefas));  
}
```

O *framework* do *Bootstrap* adiciona uma folha de estilos CSS específica, com suas próprias classes. Com sua documentação e a comunidade que a mantém atualizada, temos uma ótima alternativa de estilização sem muitos esforços,. bastando apenas, sabermos as classes a serem utilizadas, o que pode ser consultado diretamente na documentação do *Bootstrap* (BOOTSTRAP, 2021).

Agora, para finalizar o projeto, utilize seus conhecimentos para alterar as páginas e também ajustar a integração da folha de estilos criada para a página inicial com o CSS do *Bootstrap*.



## Referências

MDN Web Docs. **<div>**. 2019. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element/div>. Acesso em: 8 mar. 2021.

BOOTSTRAP. **Documentation**. 2021. Disponível em: <https://getbootstrap.com/docs/4.0/getting-started/introduction/>. Acesso em: 8 mar. 2021.

SILVA, M. S. **JavaScript-Guia do Programador**: guia completo das funcionalidades de linguagem JavaScript. São Paulo: Novatec, 2020.

W3SCHOOLS. **JavaScript and HTML DOM Reference**. 2020. Disponível em <https://www.w3schools.com/css/default.asp>. Acesso em: 8 mar. 2021.



**BONS ESTUDOS!**

---