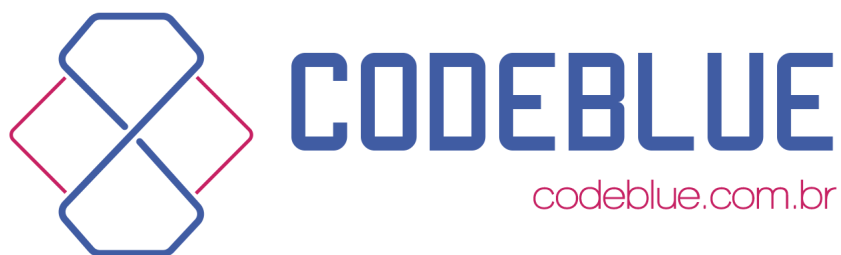


CODEBLUE



**DESENVOLVEDOR
FRONT-END**



É expressamente proibida a reprodução, divulgação, alteração ou uso não autorizado deste material. Ele é de uso exclusivo para alunos CodeBlue que adquiriram o curso de Desenvolvedor Front-end. Não se atentar a este aviso pode ocasionar problemas legais com base na lei de direitos autorais.



CAPÍTULO 1

INTRODUÇÃO AO **WEB DESIGN**

Capítulo 1 - Introdução ao web design

A internet é um meio de comunicação. Dentre as muitas possibilidades que este meio de comunicação oferece, como apps e sistemas, iremos focar aqui em páginas interativas, conhecidas como *sites*. Estas são algumas classificações que podem ser dadas à *websites*:

- **Hotsite** – *Site* pequeno e pontual, geralmente sobre uma promoção ou campanha;
- **Landing Page** – Uma única página que foge à identidade visual do resto do projeto. Por exemplo, um *site* que possui uma *Landing Page* mais bonita e criativa sobre determinado produto;
- **One Page** – Um *site* de página única, onde o menu apenas muda a posição da barra de rolagem do usuário;
- **Site Institucional** – Um *site* de uma empresa ou organização que apresenta sua ideia, aponta para as redes sociais e traz novidades;
- **Blog** – Um *site* textual, onde a pessoa ou empresa faz posts, pequenos textos, sobre assuntos que lhe dizem respeito;
- **E-commerce** – Uma loja *online*;
- **Portal** – Geralmente utilizado para trazer notícias.

Não importa qual tipo de *site* estamos falando, eles geralmente irão ser acessados através de um programa chamado navegador, ou *browser*. Este programa tem a única intenção de pegar as informações armazenadas do *site* e transcrevê-las para o usuário de forma interativa.

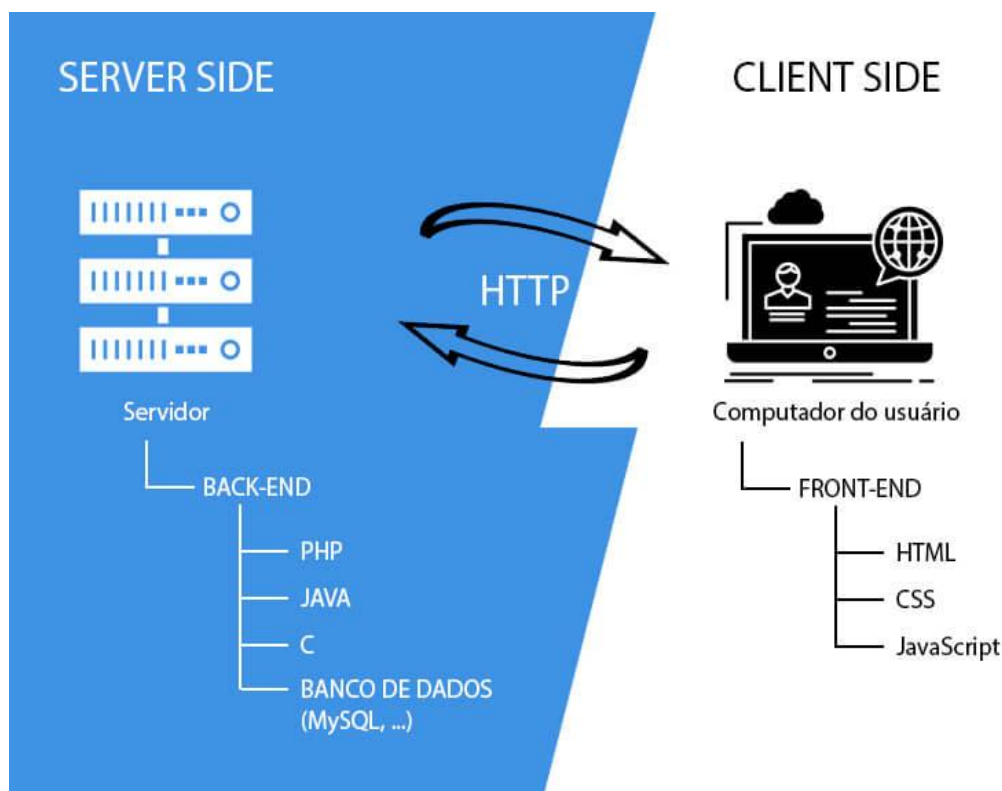
1.1.1 As Webstandards

Para coordenar e padronizar as linguagens de marcação, em 1994, Tim Berners-Lee em parceria com o CERN cria o W3C (*World Wide Web Consortium*), com sede no MIT (*Massachusetts Institute of Technology*), um consórcio internacional formado por em média 400 membros entre empresas, órgãos governamentais, instituições e pesquisadores. Esses padrões são o que chamamos de *Webstandards*, que nos dizem a forma correta de utilizar as linguagens *web* até os dias de hoje.

Veja que padronizar de forma alguma busca limitar ou formalizar a criatividade, mas sim tornar as linguagens *web* possíveis de serem utilizadas por todo o mundo, em diversos navegadores. É dever de todos os designers e desenvolvedores *web* sempre buscar enquadrar seus projetos dentro dos padrões mundiais, para que dessa forma manter, atualizar e administrar os mesmos não se torne uma tarefa árdua e complicada.

1.2 O que é Front-end?

A tecnologia *web* atual pode ser traduzida da seguinte forma:



Quando um usuário acessa um site na internet através de seu navegador, este *site* está hospedado em um servidor (*Host*). Através de um protocolo **HTTP**, o servidor onde o *site* está hospedado é consultado, e retorna para o usuário os arquivos necessários para renderizar o *site* solicitado no navegador. Estes arquivos que são entregues para o usuário dizemos que estão do lado do cliente (*client side*), e formam o *front-end* do *site*. Outros arquivos se mantêm sempre funcionando no lado do servidor (*server side*), e formam a camada de *back-end* do *site*. Aqui iremos focar em tudo o que é relacionado ao *front-end* de *sites* e aplicações.

1.2.1 Funcionamento do Front-end

Como vimos, *Front-end* pode ser traduzido como o nome dado para os arquivos que são renderizados no lado do cliente. Estes arquivos são frequentemente dos seguintes tipos:

Camada	Descrição
HTML (<i>HyperText Markup Language</i>)	A linguagem de marcação de hipertexto são os arquivos que trazem os conteúdos das páginas, como textos, imagens, tabelas e outros.
CSS (<i>Cascading Style Sheets</i>)	As folhas de estilo em cascata são os arquivos de estilização do <i>site</i> , como formatação de textos, cores, estrutura do <i>layout</i> e outros.
JS (<i>JavaScript</i>)	Os arquivos <i>JavaScript</i> são aqueles usados para dar comportamentos às páginas, como janelas e cálculos que a página executa.
Imagens	As imagens do <i>site</i> , geralmente em formatos como JPG, PNG ou GIF.
Fontes	As fontes dos textos do <i>site</i> .

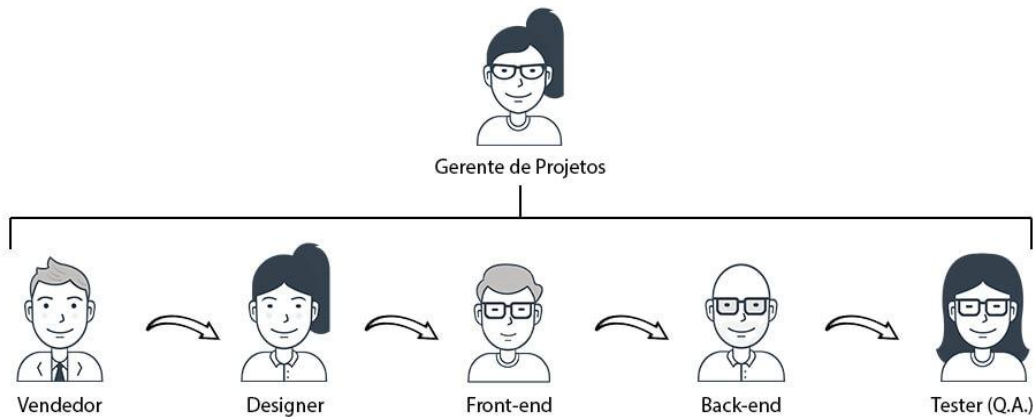
Esses tipos de arquivos também são chamados de camadas de desenvolvimento, e para criar um *site* completo precisaremos desenvolver em todas as camadas, ou seja, precisaremos criar os conteúdos em **HTML**, os estilos em **CSS**, os comportamentos em **JS**, além de consumir arquivos como imagens e fontes. Para melhor entendermos, vamos traçar um paralelo com o corpo humano, imagine que ossos e músculos são nossa estrutura (**HTML**), pele e tudo que está externo é nosso estilo (**CSS**) e sistemas como nosso sistema circulatório, digestivo ou sistema nervoso são nossos comportamentos (**JS**), e unindo todas estas camadas temos nosso corpo. Um *site* funciona da mesma forma, o **HTML**, **CSS** e **JS** são renderizados pelo nosso navegador e assim vemos o site no nosso computador.

1.2.2 O processo de hospedagem de um *site*

Nós como desenvolvedores atuamos na confecção do *site*, fazendo em nosso computador todo o código do *site* e garantindo que ele funcione perfeitamente localmente. Após esse processo, normalmente precisaremos da contratação de um servidor, que geralmente tem um custo mensal, e é onde o site será hospedado para que todos possam acessá-lo através da internet. Após a contratação, geralmente feita pelo cliente, e recebimento dos acessos, o desenvolvedor realiza um processo chamado de **FTP** (*File Transfer Protocol*), onde ele envia para o servidor todo o *site* que ele construiu localmente.

1.2.3 Qual é o papel do Front-end em um fluxo de projeto?

Antigamente tínhamos a figura do *Web Designer*, um único profissional que criava o *site* inteiro. Hoje não é mais assim. A confecção de um *site* hoje em dia funciona mais como uma linha de montagem, onde cada profissional faz uma parte do projeto e ao fim o projeto é entregue. Esta “linha de montagem” pode ser exemplificada assim:



O **Vendedor** traz os dados do projeto, como o *briefing* e referências apresentadas pelo cliente. O **Designer** desenha as telas do *site* em softwares especializados com base nas informações trazidas pelo **Vendedor**. O **Front-end** pega estas telas e converte as mesmas em **HTML**, **CSS** e **JS**, para que sejam renderizadas pelo navegador e se tornem interativas. O desenvolvedor **Back-end** faz outros sistemas que o *site* possa precisar, como formulários. O **Tester** valida se todas as solicitações do cliente com relação ao projeto foram cumpridas. E por fim, o **Gerente de Projetos** orchestra os projetos para que todos sempre tenham projetos e prioridades.

Como podemos observar, o desenvolvedor **Front-end** fica no meio do caminho, entre o **Designer** e o **Back-end**, isso significa que ele precisa, além dos conhecimentos técnicos necessários para a sua área, saber um pouco do trabalho dos outros profissionais para garantir uma boa entrega.

1.2.4 Carreira em Front-end

O desenvolvedor *front-end* pode ter uma carreira muito promissora, desde que se dedique na criação de códigos de qualidade. Geralmente a carreira passa pelos seguintes estágios:

- **Estagiário** - O desenvolvedor que está aprendendo como funciona este universo e testando seus conhecimentos;
- **Júnior** - O desenvolvedor que já consegue entregar alguns projetos mais básicos, com ajuda e supervisão de outro desenvolvedor;
- **Pleno** - O desenvolvedor que consegue entregar projetos mais complexos, sem necessidade de supervisão;
- **Sênior** - O desenvolvedor que é totalmente autogerenciável, entrega projetos complexos e tem visão de negócio, entendendo todas as etapas do que está sendo desenvolvido.

Independentemente do nível, ser um desenvolvedor exige um estudo constante de novas tecnologias.

1.3 Boas Práticas

Além de todos os conhecimentos técnicos necessários para o desenvolvimento de um *site*, que iremos ver neste curso, o bom desenvolvedor **Front-end** também precisa estar atento às diversas boas práticas para garantir que o projeto seja entregue com qualidade. Ou seja, é importante ter uma mentalidade onde o importante não é simplesmente entregar o mais rápido possível e funcionando, mas sim se atentando a diversos pontos que interferem na qualidade do código. Vamos à estas práticas:

1.3.1 UX

UX é a sigla para *User Experience*, ou Experiência do Usuário. Basicamente, consiste em utilizar práticas em seu código que melhorem a experiência de navegação do usuário, como cores de borda, espaçamentos, comportamentos de clique e afins.

1.3.2 SEO

SEO é a sigla para *Search Engine Optimization*, ou otimização para dispositivos de busca. Em resumo se trata de um conjunto de técnicas e cuidados que precisamos ter com o código para que o *site*, após ir pro ar, se posicione bem no Google.

1.3.3 Performance

Algumas técnicas e cuidados nós tomamos mais por questão de deixar o *site* mais rápido, isso garante que o usuário ao estar navegando no site, tanto no computador quanto no celular, tenha uma boa experiência de carregamento das páginas.

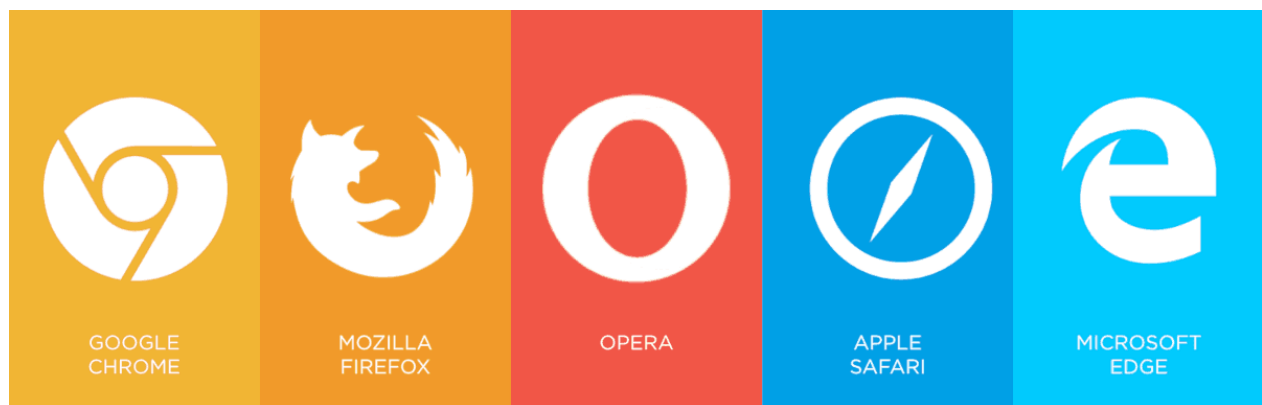
1.3.4 Semântica

Semântica na língua portuguesa é o estudo do significado das palavras. Na *web*, chamamos de semântica a utilização das *tags* e comandos para aquilo que realmente servem. É possível, mas não é semântico, utilizar comandos para uma função, sendo que na verdade ela foi criada para outra.

1.3.5 Acessibilidade

O bom profissional *web* desenvolve seus projetos levando em conta a acessibilidade. Acessibilidade é a preocupação com que o projeto possa ser acessado por qualquer tipo de usuário, desde usuários comuns até usuários portadores de necessidades especiais. Para isso, é necessário que o projeto traga alguns comandos que possibilitem o acesso a estes usuários.

Também entra no fator acessibilidade o trabalho que se tem para que o projeto tenha a possibilidade de ser exibido em diversos tipos de navegadores, dispositivos e resoluções, e ainda assim manter a mesma aparência. Esse processo de testar em navegadores diferentes é chamado de *Cross-Browser*, e deve ser feito sempre que se terminar uma etapa do projeto, pois os navegadores possuem motores de renderização diferentes entre si, fazendo também a página carregar diferente em cada um deles. Encontrando diferenças, deve-se fazer o possível para tornar a página a mais semelhante possível entre os navegadores.



Para uma boa acessibilidade é importante que o bom desenvolvedor front-end fique de olho em algumas estatísticas para conseguir atender bem a maior parte dos usuários, vejamos algumas:

- Para visualizar dados atuais sobre os navegadores mais utilizados no Brasil e no mundo, acesse aqui: [StatCounter - Browser Market Share Worldwide](#) e [W3Schools - Browsers Statics](#);
- As resoluções de telas mais utilizadas pelos usuários podem ser visualizadas aqui: [StatCounter - Screen Resolution Stats](#);
- O uso de dispositivos pelos usuários pode ser visualizado aqui: [StatCounter - Desktop / Tablet / Mobile](#).

1.4 Configurando sua IDE

IDE é a sigla dada para *Integrated Development Environment*, ou Ambiente Integrado de Desenvolvimento. Em resumo, é o nome dado para o *software* que

utilizamos para desenvolver códigos. Um código poderia ser escrito até no Bloco de Notas do *Windows*, mas já foram desenvolvidas ferramentas bem mais eficazes para a confecção de códigos. Existem várias opções de IDE's no mercado, variando muito em suas funcionalidades, e ficando a critério do desenvolvedor escolher a sua favorita. Entre as mais famosas estão o *Visual Studio Code*, *Sublime Text* e o *Dreamweaver*.

Aqui estamos utilizando o *Visual Studio Code*. Ele possui funcionalidades interessantes no que diz respeito à definição de projetos e pastas, multi seleção, autocompletar e outras funcionalidades nativas que irão facilitar e tornar muito mais dinâmico o desenvolvimento.



CAPÍTULO 2

CONSTRUINDO UMA PÁGINA **HTML**

Capítulo 2 - Entendendo o HTML

As páginas *web* (sites) são sempre hospedadas em um servidor, e acessadas pelo cliente através de um endereçamento. Quando o cliente faz uma solicitação através de uma **URL**, o pacote de dados retornado poderá ser um arquivo no formato **HTML**, que é formado por *tags* e comandos que estruturam e formam o conteúdo de toda a página. Esses comandos são traduzidos pelo navegador e exibidos ao usuário no formato de imagens, textos e objetos inseridos na página. Podemos visualizar o código **HTML** de qualquer site da *web* através da opção no navegador **Exibir código-fonte**.

Existem outras formas de se escrever o código **HTML** que talvez funcionem na prática, mas estariam fora dos padrões do **W3C**. No nosso caso, estaremos aprendendo apenas a forma correta de escrever o código, dentro dos padrões internacionais.

2.1 Sintaxe da linguagem

A estruturação do **HTML** é baseada em *tags*, ou etiquetas, que são comandos que dizem ao navegador o tipo de formatação ou estrutura que deve ser criada na página, como tabelas, parágrafos, cabeçalhos e uma infinidade de outras funções. Toda *tag* é iniciada por um sinal de menor (<) e encerrada por um sinal de maior (>). Entre esses sinais teremos a descrição da *tag* em si, como por exemplo, a *tag* <p> para parágrafos, ou para imagens. Ao inserir uma *tag* o navegador a interpretará como um comando, e insere o elemento desejado.

Também existe a *tag* de fechamento, que contém uma barra (/), como em </p>. Desta forma o texto ou objeto que sofre interferência ficará entre a *tag* de abertura e a *tag* de fechamento, como em: <p>CodeBlue</p>. No exemplo, o texto está contido em um parágrafo.

Em alguns casos podem existir *tags* que não necessitam de fechamento, mas fecham em si mesmas, por nunca haver a necessidade de colocar um elemento dentro dela, como no caso de imagens (), quebras de linha (
), campos de formulário (<input />), entre outros. Neste caso, a barra inserida no final da própria *tag* de abertura é facultativa, e pode ser omitida.

As *tags* que possuem *tag* de fechamento são chamadas de **Elemento Conteúdo**, e as que fecham em si mesmas de *tag* de **Elemento Vazio**. Veja o exemplo:

Tag de elemento conteúdo	Tag de elemento vazio
Texto em parágrafo	
Texto em div	
Texto em negrito	<input type="button" value="enviar">

2.2 Atributos

Algumas *tags* precisam de mais especificações para chegarmos ao objetivo desejado. Essas especificações serão descritas dentro da própria *tag* e são chamadas de atributos. Como por exemplo, a *tag* `<img>` insere uma imagem, mas ela precisa de atributos que mostrem onde se encontra a imagem que gostaríamos de inserir, e seu nome e extensão, como no exemplo:

```

```

Note que o atributo é colocado logo depois da *tag*, mas ainda antes do sinal de maior, separado com um espaço. Note também que todo o atributo precisa de uma especificação do seu respectivo valor, e que o valor vem sempre depois de um sinal de igual e fica entre aspas duplas. Essa será a sintaxe padrão para todos os atributos de *tags*.

Existem atributos que são chamados de **atributos globais**, ou seja, podem ser inseridos em qualquer *tag* do documento. Esta lista de atributos globais é ainda maior agora com o **HTML5** (você pode conferir uma tabela completa no final da apostila). Já grande parte dos atributos só funcionam se inseridos em determinadas *tags* específicas. Vamos conhecer algumas regras para o uso correto do **HTML**:

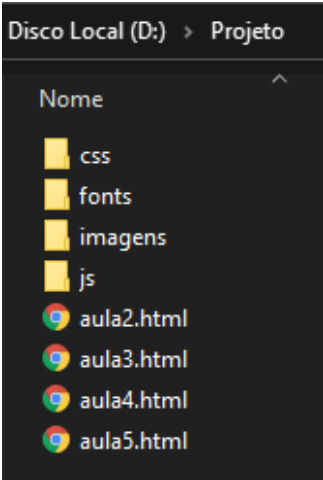
- As *tags* devem ser escritas em letras minúsculas;
- As *tags* devem ser aninhadas;
 - A primeira *tag* aberta deve ser a última a ser fechada, e a última *tag* aberta deve ser a primeira a ser fechada.
- Sempre fechar as *tags*;
- Todo o atributo deve possuir o sinal de igual, e o valor dele colocado entre aspas;
 - Há raras exceções em que o atributo não tem valor, por não haver necessidade de especificação, esses chamados de atributos *booleanos*, como em:

```
<input type="checkbox" name="cursos" checked>
```

2.3 - Criando um documento HTML

Para criar um documento HTML, em sua IDE acesse o menu *Arquivo/Salvar como...* e defina o nome do arquivo seguido da extensão `.html`. Caso esteja criando a página inicial do site, salve com o nome **index.html** para que o servidor identifique o documento como página inicial.

IMPORTANTE: Os nomes de todos os arquivos do projeto que serão publicados na *web* não podem conter espaços em branco, acentos, cedilhas ou caracteres especiais.



Dica: Crie uma pasta para armazenar todos os arquivos do seu projeto, com subpastas para cada categoria de arquivos, como imagens, CSS, JS e assim por diante.

2.4 Estrutura básica

Depois de abrir o *software*, teremos algumas *tags* padrões a serem inseridas, que devem existir em todo o documento **HTML** para podermos começar, são elas:

- DTD (*Document Type Definition*);

É a definição do tipo de documento, que diz ao navegador o tipo de *site* que estamos criando. Ele deve ser escrito na primeira linha do código, não podendo haver nada antes dele. No **HTML5** tivemos uma evolução para simplificar o **DTD**, levando em consideração que o anterior era de difícil assimilação. No **HTML5** o **DTD** ficou como:

```
<!DOCTYPE html>
```

- Tag html

Tag que dá início ao projeto. Ela é fechada ao final do projeto com `</html>`. No **HTML5** define-se o idioma que será utilizado no projeto, ficando assim:

```
<html lang="pt-br">
```

- Tag head

Define o cabeçalho da página. O que inserirmos entre a tag `<head>` e a `</head>` não entrará na página como elemento físico, mas irá influenciá-la, como no caso da tag `<title>` que veremos a seguir, *meta-tags* e *tags* de ligação com documentos externos.

- *Tag title*

O *title* ou título do *site* é o nome que aparecerá na barra de título do navegador, em suas guias, além de servir como título da página no resultado de busca do **Google**. Permanece entre as *tags* **<head>** e **</head>**. Colocaremos o título da página entre as *tags* de abertura e fechamento:

```
<title>Título</title>
```

- *Tag body*

O corpo do *site*, onde serão inseridas as informações em si. Ao final, a estrutura básica para iniciarmos qualquer projeto ficará assim:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8">
    <title>Título</title>
  </head>
  <body>

  </body>
</html>
```

Obs.: Note que os códigos foram organizados com tabulações para facilitar seu entendimento, chamamos isso de **indentação**. Não interfere na função do código, mas facilitará a leitura humana e a atualização do mesmo.



CAPÍTULO 3

CRIAÇÃO DE **TEXTOS** PARA SEU **SITE**

Capítulo 3 - Textos

Em um *site* constantemente vemos textos inseridos, mas eles não podem estar “soltos” na *tag* **body**, precisam de um container que os envolvam. Existem containers semânticos para cada tipo de conteúdo diferente, precisando de discernimento do desenvolvedor para validar o tipo de conteúdo para decidir qual *tag* utilizar. As principais *tags* para a marcação de textos são:

Tag	Função
<code></code>	Negrito.
<code></code>	Itálico.
<code><p></p></code>	Novo parágrafo.
<code>
</code>	Quebra de linha.
<code><h1></h1></code> , <code><h2></h2></code> , <code><h3></h3></code> , ...	<i>Tags</i> de Cabeçalho.
<code></code>	Estilização linear.
<code><hr></code>	Linha horizontal.

Veja o exemplo do uso das *tags* mais comuns:

```
<body>
  <p><strong>Web Standards</strong> são um conjunto de normas,
  diretrizes, recomendações, notas, artigos, tutoriais e afins de caráter
  técnico, produzidos pelo <em>W3C</em> e destinados a orientar fabricantes,
  desenvolvedores e projetistas. <br> Seu uso possibilita a criação de uma
  <em>web</em> acessível a todos, independentemente dos dispositivos usados
  ou de suas necessidades especiais.</p>
</body>
```

3.1.1 Cabeçalhos

As *tags* de cabeçalho (ou *headings*) são extremamente difundidas e utilizadas nos dias de hoje. Elas são seis: `<h1></h1>`, `<h2></h2>`, `<h3></h3>`, `<h4></h4>`, `<h5></h5>` e `<h6></h6>`. Devem ser utilizadas para serem o container semântico para títulos, por ordem hierárquica de importância, sendo o **h1** o mais importante e o **h6** o menos importante. Veja o exemplo:

```
<body>
  <h1></h1>
  <h2>Notícia 1</h2>
  <p>Lorem Ipsum is simply dummy text of the printing and typesetting
  industry. Lorem Ipsum has been the industry's standard dummy text eve
```

```

since the 1500s, when an unknown printer took a galley of type and
scrambled it to make a type specimen book.</p>
  <h2>Notícia 2</h2>
  <p>Lorem Ipsum is simply dummy text of the printing and typesetting
industry. Lorem Ipsum has been the industry's standard dummy text
ever since the 1500s, when an unknown printer took a galley of type
and scrambled it to make a type specimen book.</p>
</body>

```

Dica: É comum o uso da *tag* **h1** para envolver o logo do *site* e nome do mesmo, para melhorar os resultados nos dispositivos de busca, visto que estes levam em conta a relevância, e o título **h1** é o nível de título mais relevante, e deve ser utilizado uma única vez na página.

3.1.2 Listas

O **HTML** possui dois tipos de listas:

- Lista não ordenada, definida pela *tag* ****;
- Lista ordenada, definida pela *tag* ****.

Cada item da lista será definido pela *tag* ****. Veja um exemplo:

```

<body>
  <ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
  </ul>
</body>

```

Em alguns momentos uma lista pode possuir sublistas, veja um exemplo de como construir este código:

```

<body>
  <ul>
    <li>Front-end
      <ul>
        <li>HTML</li>
        <li>CSS</li>
        <li>JavaScript</li>
      </ul>
    </li>
    <li>Back-end
      <ul>

```

```
        <li>PHP</li>
        <li>.NET</li>
        <li>MySQL</li>
    </ul>
</li>
</ul>
</body>
```

3.2 Comentários

Para uma melhor compreensão do código inserido, é de bom grado que se comente as regiões do código correspondentes a cada conteúdo. Para comentar o código **HTML**, faremos da seguinte forma:

```
<!-- aqui inserimos o comentário -->
```

Podemos comentar qualquer área do código, não prejudicando em nada sua sintaxe ou renderização, pois o navegador sempre irá ignorá-lo.



CAPÍTULO 4

RECURSOS VISUAIS DO **SITE**

Capítulo 4 - Recursos Visuais do Site

4.1 Imagens

Imagens são um recurso visual extremamente importante em um site, mais da metade dos elementos que vemos na tela normalmente são imagens, e precisamos tomar diversos cuidados para fazer a chamada dessas imagens de forma correta. Preocupe-se com o tamanho do arquivo em pixels, para ter imagens em boa resolução, mas procure equilibrar isso com o tamanho da imagem em bytes, para não “pesar” muito o seu projeto.

Assim como as *tags* de formatação de texto, a *tag* para inserir imagens deve ser colocada na seção `<body></body>` da estrutura do código **HTML**. A *tag* para inserir imagens é a *tag* `<img>`, que é uma *tag* de elemento vazio. Este comando sozinho não fará nenhuma alteração na página, precisamos dar alguns atributos para inserirmos a imagem corretamente. São eles:

Atributos	Função
src	Determina o caminho para a imagem que gostaríamos de inserir, seguido do nome e extensão da mesma.
alt	Texto alternativo para que deficientes visuais possam saber qual objeto está inserido.
width	Largura da imagem. Valor em pixels.
height	Altura da imagem. Valor em pixels.

Veja o exemplo:

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

Sempre insira a extensão da imagem, que geralmente varia entre .jpg, .gif ou .png. Também defina largura e altura para o mesmo tamanho que a imagem já é, visto que redimensioná-la por **HTML** trará mais peso em *bytes* para a página, prejudicando seu desempenho. É necessário algum conhecimento em um *software* gráfico para redimensionar a imagem antes de inseri-la no código. Note também que é necessário definir o caminho para chegar ao arquivo de imagem. Isso irá acontecer em diversos outros momentos, como na definição de links e outros arquivos. Para definir o caminho,

define-se ../ sempre que queremos voltar um nível de pastas, e o nome da pasta seguido de uma barra sempre que gostaríamos de avançar neste caminho.

Dica: O atributo *title* é um **atributo global**. Pode ser inserido em qualquer *tag* para que o texto contido em seu valor apareça quando o usuário posicionar o ponteiro do mouse sobre o elemento.

4.1.1 Tag figure

Uma das novas *tags* do **HTML5** é a *tag* `<figure></figure>`. Ela será utilizada para envolver as imagens do seu projeto em um *container*, que pode mais tarde ser estilizado por **CSS**. Também foi lançada a *tag* `<figcaption></figcaption>` para inserir uma legenda para a imagem a ser exibida na página.

```
<body>
  <figure>
    
    <figcaption>Legenda da foto</figcaption>
  </figure>
</body>
```

4.1.2 Boas práticas para imagens

Além do que vimos, o **HTML** possui hoje mais algumas melhorias nativas possíveis de serem feitas com imagens, vejamos:

- Tag `<picture></picture>`;

Esta *tag* permite definirmos outras imagens para serem carregadas em dispositivos diferentes, para não acontecer de uma imagem pesada, pensada para computadores, também seja chamada em celulares utilizando 3G. Mais a frente aprenderemos o desenvolvimento de *layouts* responsivos e todas as suas práticas, mas esta é uma das técnicas que facilita o desenvolvimento e traz uma melhor experiência ao usuário. Estas imagens estarão em *tags* `<source>` e definidas em qual dispositivo serão carregadas através do atributo *media*.

```
<body>
  <picture>
    <source media="(min-width:650px) "
srcset="img_pink_flowers.jpg">
    <source media="(min-width:465px) "
srcset="img_white_flower.jpg">
    
  </picture>
</body>
```

- Atributo *loading*.

Antigamente utilizávamos um *plug-in JavaScript* chamado **LazyLoad** para que os itens mais pesados da página só fossem carregados no momento em que estivessem sendo exibidos na tela, melhorando assim a performance da página. Hoje temos um atributo que faz a mesma função e é nativo do **HTML**, o *loading*, utilizado para *tags* `img` que recebe o valor **lazy** quando declarado.

```

```

4.2 Favicon

Uma *tag* comumente inserida na seção **head** da página é a `<link>`. A *tag* **link** irá ligar o arquivo onde ela é inserida a um outro arquivo externo de estilização que

influenciará o documento **HTML**. Por exemplo, podemos linkar um documento em um folha de estilos **CSS** que irá estilizá-lo. Veremos com mais detalhes no capítulo de **CSS**. Outro momento que pode ser utilizada a *tag* **link** é para a inserção dos chamados *Favicons*. São aqueles pequenos ícones que ficam ao lado do título na aba da página. Para inserção de um *Favicon*, é necessário primeiramente gerar a arte da imagem que gostaríamos, de preferência na proporção 16x16 pixels, e em seguida ligar ela ao documento desejado através da tag, veja um exemplo:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8">
    <title>Elemento Link</title>
    <link rel="icon" href="imagens/favicon.png">
  </head>
  <body>

  </body>
</html>
```

4.3 Tabelas

Tabelas por padrão são utilizadas para a organização de dados tabulares, mas por muito tempo foram utilizadas para estruturação de *layouts*. Tabelas são a forma mais primitiva de se construir um *layout*. Por mais que nos dias atuais estejam em total desuso se usadas para estruturação de *layouts*, ainda hoje são utilizadas para estruturação de *mailmarketing*, páginas sem suporte a **CSS** e para inserção de dados tabulares.

A *tag* para a criação de tabelas é a **<table></table>**. Mas ela de forma isolada, apenas inicia e finaliza a região para a tabela, ainda é preciso dar-lhe propriedades e criar as células.

4.3.1 Criação dos elementos da tabela

Após a criação da tabela com sua devida formatação, precisamos inserir as regiões da tabela, formadas pelas *tags* **<thead></thead>**, **<tbody></tbody>** e **<tfoot></tfoot>**, que respectivamente separam as regiões de cabeçalho, corpo e rodapé da tabela, não sendo obrigatório sua inserção, mas uma prática muito recomendada. Após isso, devemos criar as células que irão receber o conteúdo. Para isso utilizamos as *tags* **<th></th>** para cabeçalho da tabela, ou seja, para que a informação contida na célula receba a formatação de negrito e centralizado, **<tr></tr>** para a criação de linhas e **<td></td>** para a criação de colunas. Ao fim, a tabela criada ficará como no exemplo:

```

<body>
  <table>
    <thead>
      <tr>
        <th scope="col">Carros</th>
        <th scope="col">Valores</th>
        <th scope="col">Ano</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>Gol</td>
        <td>R$ 28.000,00</td>
        <td>2007</td>
      </tr>
      <tr>
        <td>Corsa</td>
        <td>R$ 15.000,00</td>
        <td>2003</td>
      </tr>
      <tr>
        <td>Uno</td>
        <td>R$ 25.000,00</td>
        <td>2013</td>
      </tr>
      <tr>
        <td>Celta</td>
        <td>R$ 17.000,00</td>
        <td>2016</td>
      </tr>
    </tbody>
  </table>
</body>

```

Note que o conteúdo da célula, seja um texto, uma imagem ou outro elemento fica entre as *tags* de abertura e fechamento da célula. No caso da tabela exemplo, não foi utilizada a *tag* **<tfoot>**, visto que essa tabela em específico não possui rodapé. Um exemplo de uma tabela com rodapé seria uma tabela de valores com uma última linha para totais.

O atributo *scope* é permitido para a *tag* **<th>**, e define por questão de acessibilidade ao que se referencia aquele cabeçalho.

Obs.: Mesmo que você não utilize mais tabelas para a estruturação do *layout*, certamente pode criá-las para organizar informações do *site*, como tabelas de preços, por exemplo.

Caso haja a necessidade, pode-se usar os atributos **colspan**, para mesclar colunas, ou **rowspan**, para mesclar linhas. O valor do atributo deverá ser a quantidade de linhas ou colunas que gostaríamos de mesclar.

4.4 Iframes

Iframes são regiões ou seções criadas na página que exibem parte ou todo um conteúdo dentro de uma região específica. Por exemplo, poderíamos ter uma região inserida no nosso site onde é exibido parte do site da **CodeBlue**, sendo como uma janela que nos traz diretamente a informação contida no site, não dependendo do local onde está inserido. Seguindo esta ideia, poderíamos inserir **iframes** no nosso site como a cotação do dólar ou previsão do tempo, por exemplo, sem a necessidade de atualizarmos esta informação. Veja:

```
<iframe src="http://www.codeblue.com.br"></iframe>
```

Iframes também podem receber o atributo **loading="lazy"**, como as imagens.

4.5 Áudio

Para a inserção de áudio em seu site por **HTML5** utilizamos a *tag* **<audio></audio>**:

```
<audio src="som.mp3"></audio>
```

A *tag* **audio** pode receber os seguintes atributos:

Atributos	Função
src	Define a música a ser tocada.
autoplay	Atributo <i>booleano</i> . Define que a música será tocada assim que estiver pronta.
controls	Atributo <i>booleano</i> . Controles serão mostrados.
loop	Atributo <i>booleano</i> . A música tocará novamente assim que terminar.
preload	Atributo <i>booleano</i> . O áudio será primeiro totalmente carregado para depois ser tocado. É cancelado com atributo autoplay.

Veja um exemplo:

```
<audio src="som.mp3" autoplay loop controls></audio>
```

Caso queiramos inserir formatos alternativos para o som, para não acontecer de termos problemas de suporte de leitura do mesmo pelo navegador, podemos utilizar a *tag* **<source>** dentro da *tag* **<audio></audio>**:

```
<audio autoplay controls>
  <source src="som.mp3" type="audio/mpeg">
  <source src="som.wav" type="audio/wave">
  <source src="som.ogg" type="audio/ogg">
</audio>
```

A *tag* **source** possui os seguintes atributos e valores:

Atributos	Valores
src	Define a música a ser tocada.
type	Define o tipo do arquivo.

O tipo de *player* utilizado dependerá do navegador em que o *site* é aberto. É possível personalizá-lo através do uso de **JavaScript**.

4.6 Vídeo

Para inserirmos vídeos por **HTML5** utilizaremos a *tag* **<video></video>**:

```
<video src="video.mp4"></video>
```

As extensões da *tag* **video** são semelhantes as da *tag* **audio**, em suas propriedades e valores, assim como o uso da *tag* filha **source**:

```
<video autoplay controls>
  <source src="video.mp4" type="video/mp4">
  <source src="video.ogv" type="video/ogg">
  <source src="video.webm" type="video/webm">
</video>
```

É possível utilizar os atributos da *tag* **audio** vistos anteriormente e ainda outros atributos na *tag* **video** para alguns controles adicionais:

Atributos	Valores
poster	Nome e extensão de uma imagem para ser definida como capa do vídeo enquanto ele ainda estiver pausado ou carregando.
audio	Valor <i>muted</i> para que o vídeo seja isento de áudio.
width	Valor numérico para largura do <i>player</i> . Caso as dimensões sejam diferentes do <i>aspect ratio</i> do vídeo, o mesmo será preservado.
height	Valor numérico para altura do <i>player</i> . Caso as dimensões sejam diferentes do <i>aspect ratio</i> do vídeo, o mesmo será preservado.

O tipo de *player* utilizado dependerá do navegador em que o *site* é aberto. É possível personalizar controles através do uso de **JavaScript**. Veja um exemplo de um áudio e um vídeo incorporado a um *site*:

```
<body>
  <audio preload loop controls>
    <source src="som.mp3" type="audio/mpeg">
    <source src="som.wav" type="audio/wave">
    <source src="som.ogg" type="audio/ogg">
  </audio>
  <video autoplay controls width="500" height="350">
    <source src="video.mp4" type="video/mp4">
    <source src="video.ogv" type="video/ogg">
    <source src="video.webm" type="video/webm">
  </video>
</body>
```



CAPÍTULO 5

LINKS E **META-TAGS**

Capítulo 5 - Links

Link significa ligação. Irá ligar um elemento inserido no *site*, como um texto, uma imagem ou um objeto à página que deve ser aberta quando o usuário interagir com o objeto. Podemos inserir *links* em qualquer objeto do *site*, e para isso utilizaremos a *tag* `<a></a>`, ficando o elemento *clicável* entre as *tags* de abertura e fechamento. Essa *tag* sozinha não fará efeito algum, ela precisará de alguns atributos para ter sua função especificada. Veja:

Atributos	Função
href	Principal atributo que define o local para onde o usuário será enviado depois de interagir com o objeto <i>clicável</i> .
target	Define em qual janela o <i>link</i> deve ser aberto. Os valores podem variar em: _self , para o <i>link</i> ser aberto na mesma janela, _blank , para o <i>link</i> ser aberto em outra janela.
name	Utilizado em âncoras.

Veja o exemplo:

```
<a href="http://www.codeblue.com.br" target="_blank">Clique aqui</a>
```

Note que ao se definir *link*, o texto referente ao *link* recebe automaticamente uma formatação padrão de cor e estilo, que poderá ser alterada mais tarde pelas **CSS**.

5.1.1 Tipos de links

Dos tipos de *link* existentes, veremos os quatro mais comuns nos dias de hoje:

- *Link* Relativo;

Um *link* que tem como página de destino uma página de navegação interna do *site*. Para criá-lo, damos como valor do atributo **href** o caminho e o nome do arquivo desejado. Exemplo:

```
<a href="contato.html">Contato</a>
```

- *Link Absoluto;*

Link para uma página externa ao site. Neste caso, como valor do atributo **href** damos o endereço completo da página de destino. Exemplo:

```
<a href="http://www.codeblue.com.br" target="_blank">CodeBlue</a>
```

Em geral para esses casos damos ao atributo **target** o valor **_blank**, para que a página de destino seja aberta em outra janela e o usuário continue com a navegação no site atual.

- *Link de download;*

Link para fazer o *download* de um arquivo. Semelhante ao *link* relativo, mas como valor do atributo **href** temos um arquivo que não contém uma extensão **.htm** ou **.html**, mas alguma outra não compreendida pelo navegador. Exemplo:

```
<a href="musica.zip" title="Faça o Download da Música">Download</a>
```

Neste caso não precisamos definir **target**, já que a página não será aberta, mas sim será destinado o *download* do arquivo ao usuário.

- *Link de âncora.*

Uma **âncora** é um *link* para a mesma página, mas em outro ponto da barra de rolagem, para casos como artigos extensos onde ao final poderíamos encontrar um *link* que já nos enviaria de volta para o topo da página. Para isso, precisaremos do atributo **id**. Ele irá marcar o ponto de retorno na barra de rolagem quando o usuário clicar no *link*. Veja o exemplo:

```
<body>
  <a id="topo"></a>
  <br> <br> <br> <br> <br> <br> <br> <br> <br> <br> <br> <br> <br>
  <br> <br> <br> <br> <br> <br> <br> <br> <br> <br> <br> <br>
  <a href="#topo">TOPO</a>
</body>
```

O valor do atributo **id** pode ser qualquer nome, sem espaço ou caracteres especiais. Este nome é referenciado no *link* ao final do documento, precedido por uma cerquilha (#), e assim o navegador compreende que é um *link* para a mesma página, fazendo a ligação entre o valor do atributo **id** no início com o valor do atributo **href** ao fim do documento.

5.2 Meta-tags

As meta-tags são *tags* inseridas na seção `<head></head>` da página, e tem o intuito não de inserir um conteúdo propriamente dito, mas sim inserir valores, que influenciam a página. As *tags* `<meta>` contém os atributos como **name**, para especificação do tipo de *meta-tag*, e **content**, que informa o conteúdo da *meta-tag*. Conheça as principais:

5.2.1 Meta-tags relacionadas a SEO

SEO (*Search Engine Optimization* ou otimização para dispositivos de busca) são técnicas utilizadas para melhorar o resultado do seu projeto nos dispositivos de busca, melhorando assim também seu retorno comercial. É muito necessário hoje ter o seu projeto otimizado, e uma das etapas é declarar corretamente as *meta-tags* lidas por esses dispositivos, são elas:

- Description;

Informa o texto de descrição que aparecerá logo abaixo de um resultado em um dispositivo de busca.

```
<meta name="description" content="A primeira agência para agências do Brasil">
```

- Robots;

Informa aos dispositivos de busca como o **Google**, que é permitida a indexação dessa página e das páginas que a ela estiverem ligadas, com os valores *"index, follow"* ou *"all"*. É também possível solicitar o efeito contrário, que os dispositivos de busca não indexem a página e nem as que se seguem, com a declaração do valor *"noindex, nofollow"*.

```
<meta name="robots" content="index, follow">
```

- Keywords.

Informa palavras-chave que funcionarão como referência para os dispositivos de busca. Separam-se os termos por vírgula. O **Google** não olha mais para esse *tag*, então ela está em desuso, só fazendo sentido caso queira otimizar para outros buscadores.

```
<meta name="keywords" content="html, css, js">
```

5.2.2 Meta-tags de descrição do projeto

- Rating;

Define uma classificação etária para o *site*. Os valores do atributo **content** podem variar entre *general*, para qualquer idade, *14 years*, por exemplo, para censura de 14 anos ou *mature*, para maiores de idade. Alguns navegadores utilizam esta *meta-tag* como parâmetro para filtro.

```
<meta name="rating" content="general">
```

- Charset.

Utilizaremos a *meta-tag* para a definição do tipo de caracteres, padrão **W3C** para esse tipo de documento. É necessário sempre definir o tipo de caracteres no documento para que o navegador possa interpretar os caracteres utilizados. Por padrão utiliza-se utf-8, mas podem haver variações. Caso não seja definido, deve-se utilizar um código que gere o caractere desejado, confira a tabela completa desses caracteres no capítulo **Tabelas de referência**.

```
<meta charset="utf-8">
```

Exemplo de aplicação das *meta-tags*:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8">
    <meta name="description" content="A primeira agência para
agências do Brasil">
    <meta name="robots" content="index, follow">
    <meta name="rating" content="general">
    <title>Aprendendo meta-tags</title>
  </head>
  <body>

  </body>
</html>
```



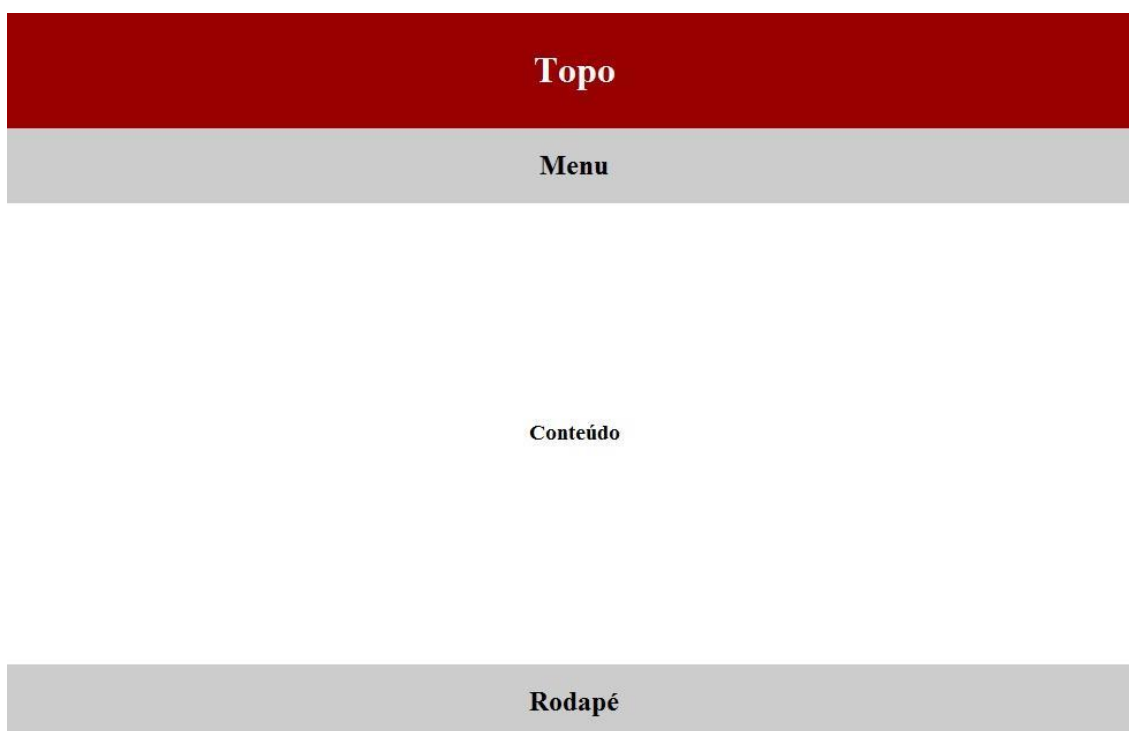
CAPÍTULO 6

ESTRUTURAÇÃO DE **LAYOUT**

Capítulo 6 - Estruturando o layout

Layout é uma palavra que significa formato. No caso das páginas **HTML** significará a forma em que o site está estruturado. Você já deve ter percebido que inserir o conteúdo no site é plenamente viável, mas organizá-lo de forma a ter a aparência de uma página *web* é o que faremos agora. Este processo pode ser feito de várias formas: através de tabelas, de **div's**, **iframes** ou pelas novas *tags* **HTML5**.

Um *Layout* é formado por algumas partes essenciais para a criação de um projeto web, algumas dessas partes devem permanecer constantes, como o topo, menu e rodapé, e outras mutáveis, como as áreas de conteúdo.



6.1 Layout por divs

Div é uma *tag* que cria uma seção na página para inserirmos qualquer tipo de conteúdo, por isso é uma *tag* livre de semântica. Ela sozinha não poderá fazer muita coisa. Precisaremos de um nível mais avançado de **CSS** para criarmos um *layout* com a *tag* `<div></div>`. Por hora, veremos algumas propriedades.

Uma **div** é como uma célula de tabela flutuante, onde inserimos o conteúdo desejado entre a *tag* de abertura e fechamento. Um *layout* por **div** tem maior facilidade de atualização, maior dinamismo na estilização com folhas de estilo e uma variação de efeitos mais abrangente. Por exemplo, por **CSS3** hoje podemos arredondar os cantos de uma **div** ou colocar efeitos de sombreamento.

Nota: Também utilizamos a tag `<span></span>` para linhas de conteúdo, mas neste caso para conteúdos em linha.

Para a estilização das **div's** por **CSS** precisaremos colocar o atributo **class** dentro de cada uma, para identificar e em seguida estilizar. Por padrão, geralmente criamos as seguintes **classes** para as **div's** do *layout*:

Div wrap

Div header

Div navbar

Div content

Div footer

- A **div wrap** é aquela que vai conter todos os outros elementos, a que será centralizada e aplicada algum efeito de sombra, por exemplo;
- A **div header** é o espaço destinado ao topo da página;
- A **div navbar** é a barra de navegação do site, como no caso de um menu horizontal;
- A **div content** traz o conteúdo da página em questão. Normalmente conterá outras **div's** para separar o conteúdo;
- Criamos a **div footer** para o rodapé da página.

Obs.: Você pode dar qualquer valor para o atributo **class**, esses valores aqui colocados são apenas senso comum.

O código **HTML** para esta página é simples, pois apenas iremos inserir e identificar os objetos, serão as **CSS** que dirão seu tamanho, cor, alinhamento, posição e estilo. Veja o código **HTML** da página:

```
<body>
  <div class="wrap">
    <!-- Aqui começa o topo -->
```

```

<div class="header">
  <h1> Topo </h1>
</div>
<!-- Aqui começa o menu -->
<div class="navbar">
  <ul>
    <li><a href="index.html">Home</a></li>
    <li><a href="sobre.html">Sobre</a></li>
    <li><a href="produtos.html">Produtos</a></li>
    <li><a href="contato.html">Contato</a></li>
  </ul>
</div>
<!-- Aqui começa o conteúdo -->
<div class="content">
  <div class="esquerda">
    <h2>Notícia 1</h2>
    <p>O Consórcio World Wide Web (W3C) anunciou o
lançamento de mais um escritório internacional, agora na Rússia. como
resultado da crescente inserção e participação global no Consórcio, O
Escritório está hospedado na Escola Superior de Economia (HSE) da
Universidade Nacional de Pesquisa, fundada em 1992...</p>
    <p><a
href="http://www.w3c.br/Noticias/W3cAbreEscritorioNaRussia">leia
mais...</a></p>
  </div>
  <div class="centro">
    <h2>Notícia 2</h2>
    <p>Esta oitava edição do Prêmio Mario Covas conta com
a parceria do W3C Brasil na categoria "Governo Aberto". O Prêmio
Mário Covas é uma iniciativa da Secretaria de Estado de Gestão
Pública de São Paulo e a participação do W3C Brasil inova por
introduzir o reconhecimento de iniciativas de cidadãos que utilizam
dados governamentais...</p>
    <p><a
href="http://www.w3c.br/Noticias/OitavaEdicaoPremioMarioCovas">leia
mais...</a></p>
  </div>
  <div class="direita">
    <h2>Notícia 3</h2>
    <p>O Consórcio World Wide Web (W3C) é um consórcio
internacional no qual organizações filiadas, uma equipe em tempo
integral e o público trabalham juntos para desenvolver padrões para a
Web...</p>
    <p><a href="http://www.w3c.br/Sobre">leia
mais...</a></p>
  </div>
</div>
<!-- Aqui começa o rodapé -->
<div class="footer">
  <p>&copy; Copyright - Todos os direitos reservados</p>
</div>
</div>
</body>

```

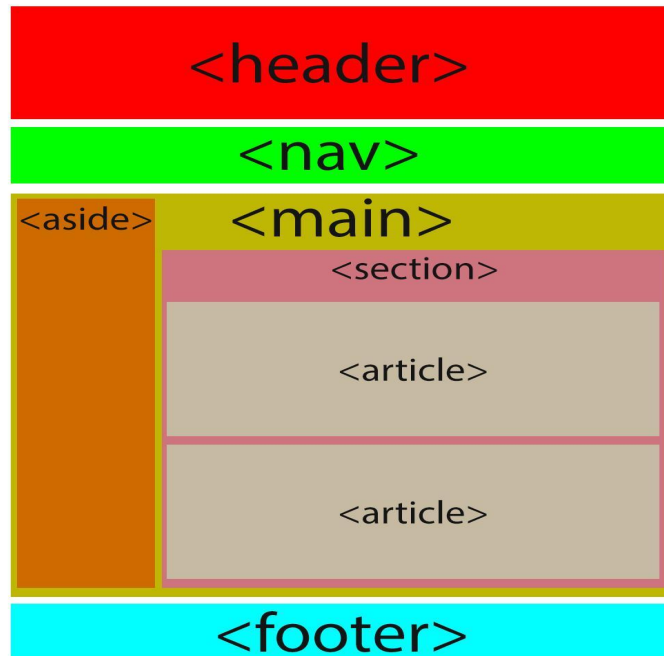
Voltaremos a abordar este mesmo *layout* no final do capítulo **Entendendo as CSS**, portanto salve este arquivo como **estrutura_div.html** para que futuramente possamos voltar a desenvolver o *layout* por completo, uma vez tendo já adquirido nosso conhecimento com folhas de estilo.

Todas as **div's** receberam **classes** para futura estilização. As **div's** esquerda, centro e direita são para a digitação do conteúdo em três colunas. Na área do menu foram inseridas *tags* para uma lista não ordenada onde faremos a estilização do mesmo. Os demais conteúdos receberam *tags* de cabeçalho e parágrafos.

Obs.: Ainda no **HTML5** é comum a utilização de *div's* para outros fins, mas para estruturação de *layout* teremos *tags* específicas para isso.

6.2 Layout pelas novas tags HTML5

O **HTML5** criou novas *tags* para estruturação de *layout*, como a *tag* `<header></header>`, `<footer></footer>`, `<nav></nav>` e assim por diante. Desta forma, a própria *tag* inserida já possui formatações específicas e ideais para cada área do *layout*. Veja o exemplo:



Agora veja o código para formar uma página como essa, já com alguns conteúdos inseridos:

```

<body>
  <!-- Aqui começa o topo -->
  <header>
    <h1>Topo</h1>
    <h2>Slogan</h2>
  </header>
  <!-- Aqui começa o menu -->
  <nav>
    <ul>
      <li><a href="index.html">Home</a></li>
      <li><a href="sobre.html">Sobre</a></li>
      <li><a href="portfolio.html">Portfólio</a></li>
      <li><a href="localizacao.html">Localização</a></li>
      <li><a href="contato.html">Contato</a></li>
    </ul>
  </nav>
  <!-- Aqui começa o conteúdo -->
  <main>
    <aside>
      <h3>Links úteis</h3>
    </aside>
    <section>
      <article>
        <h3>Notícia 1</h3>
        <p>Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos, e vem sendo utilizado desde o século XVI, quando um impressor desconhecido pegou uma bandeja de tipos e os embaralhou para fazer um livro de modelos de tipos. Lorem Ipsum sobreviveu não só a cinco séculos, como também ao salto para a editoração eletrônica, permanecendo essencialmente inalterado. Se popularizou na década de 60, quando a Letraset lançou decalques contendo passagens de Lorem Ipsum, e mais recentemente quando passou a ser integrado a softwares de editoração eletrônica como Aldus PageMaker...</p>
        <p class="leia"><a href="noticia1.html">leia mais...</a></p>
      </article>
      <article>
        <h3>Notícia 1</h3>
        <p>Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos, e vem sendo utilizado desde o século XVI, quando um impressor desconhecido pegou uma bandeja de tipos e os embaralhou para fazer um livro de modelos de tipos. Lorem Ipsum sobreviveu não só a cinco séculos, como também ao salto para a editoração eletrônica, permanecendo essencialmente inalterado. Se popularizou na década de 60, quando a Letraset lançou decalques contendo passagens de Lorem Ipsum, e mais recentemente quando passou a ser integrado a softwares de editoração eletrônica como Aldus PageMaker...</p>
        <p class="leia"><a href="noticia1.html">leia mais...</a></p>
      </article>
    </section>
  </main>

```

```
<!-- Aqui começa o rodapé -->
<footer>
  <p>&copy; Copyright - Todos os direitos reservados -
Desenvolvido por: CodeBlue</p>
</footer>
</body>
```

Salve este projeto de *layout* como **layouthtml5.html** para que futuramente possamos estilizá-lo através das **CSS** e façamos as demais páginas do projeto. Esta é apenas uma sugestão comum de uso, as tags de estruturação de *layout* do **HTML5** podem ser utilizadas em qualquer local do documento.

- A tag **header** define uma seção para o topo da página. Todo o conteúdo relativo ao topo deve ser inserido entre a tag de abertura e fechamento;
- A tag **nav** é voltada para a barra de navegação ou menu;
- A tag **main** cria uma seção para todo o conteúdo;
- A tag **section** cria um novo bloco de conteúdo;
- A tag **article** define uma área para inserção de notícias ou uma subregião de conteúdo;
- A tag **aside** é opcional para criar um sidebar, como um segundo menu de categorias na página;
- A tag **footer** cria uma seção para a inserção do rodapé da página.

O **W3C** em conjunto com os membros desenvolvedores do **HTML5** preocuparam-se em criar *tags* que facilitam a criação e manutenção da página, além de padronizar todas elas para uma mesma estrutura de *layout*. Elas serão estilizadas pelas **CSS** para estilização do seu tamanho, cor, estilo e posicionamento. Diferentemente das **div's** que são isentas de semântica, as novas *tags* **HTML5** têm a função de criar cada seção específica na estruturação do *layout*. Ainda podemos utilizar **div's** nesses *layouts*, como para inserção de elementos flutuantes, banners e propagandas, e ainda precisaremos da **div wrap** para conter todo o *layout* e centralizarmos futuramente o projeto.

Dica: As novas tags **HTML5** são como as tradicionais, deverão receber estilização por **CSS**, e insere-se seus conteúdos entre a abertura e fechamento.



CAPÍTULO 7

FORMULÁRIOS

Capítulo 7 - Trabalhando com formulários

Sempre que quisermos que o usuário possa interagir com a página *web*, preenchendo um formulário de contato, criando um cadastro ou fazendo um login precisaremos das *tags* de criação de formulários para este fim.

A *tag* para criação de formulários é a *tag* `<form></form>`. Todos os elementos do formulário deverão ficar entre as *tags* de abertura e fechamento.

```
<form action="form.php" method="post">
  <p>
    <label for="nome">Nome:</label>
    <input type="text" name="nome" id="nome">
  </p>
</form>
```

Como podemos visualizar, os demais elementos filhos estarão sempre entre o início e fim de `<form></form>`. A *tag* para início de formulário deve ter alguns atributos para um bom funcionamento. São eles:

Atributos	Função
action	O valor desse atributo será o arquivo executado quando o usuário clicar no botão de envio (<i>submit</i>).
enctype	Define o tipo de dados. Os valores variam entre application/x-www-form-urlencoded como padrão, e multipart/form-data para formulários com campos para <i>upload</i> de arquivos.
method	Método de envio das informações, podendo ter como valores GET ou POST , sendo mais comumente utilizado o atributo POST , pois ele encapsula os dados e envia.

7.1 Principais campos de formulário

A *tag* que irá inserir grande parte dos objetos de formulário é a `<input>`, mudando apenas o tipo de campo através do atributo **type**. Trata-se de uma *tag* de elemento vazio. Conheça algumas variações de campos **input**:

- Text;

Um dos campos mais utilizados. Cria um campo de texto padrão para valores genéricos. O atributo **name** define o nome do campo para que o **Back-end** possa reconhecê-lo. O atributo **type** define o tipo de **input**. O atributo **maxlength** define o número máximo de caracteres. O atributo *booleano* **disabled** desativa o campo.

```
<input type="text" name="nome" maxlength="20">
```

- Checkbox;

Para criarmos caixas onde é possível escolher mais de uma opção. O atributo **value** indica o valor que será processado. O atributo **checked** serve para que se tenha um estado inicial como marcado.

```
<input type="checkbox" name="cursos" value="html5" checked>
```

- Radio;

Botão onde apenas uma opção pode ser escolhida. Para que o efeito funcione, todos os botões da mesma categoria devem ter o mesmo valor do atributo **name**, variando apenas em **value**.

```
<input type="radio" name="genero" value="masculino">
```

- Tag `<select></select>`;

Para definirmos uma lista de opções. As opções serão definidas pela tag `<option></option>`. O atributo *booleano* **selected** é usado para uma tag `<option>` iniciar no estado de selecionada.

```
<select name="cursos">
  <option value="não selecionado" selected>Escolha um curso:
</option>
  <option value="XHTML">XHTML</option>
  <option value="CSS">CSS</option>
  <option value="PHP">PHP</option>
</select>
```

- Tag `<textarea></textarea>`.

Cria um campo de texto com mais linhas de preenchimento, para áreas como comentários, por exemplo.

```
<textarea name="mensagem"></textarea>
```

Obs.: Não se esqueça de sempre definir `name` e `value` (quando aplicável) como atributos de campos de formulário, para que o back-end possa processá-lo corretamente.

7.2 Organização semântica do formulário

Algumas tags devem ser utilizadas para organizar melhor um formulário e melhorar sua usabilidade (UX), mesmo que não insiram nenhum campo novo ao mesmo.

- Tag `<fieldset></fieldset>`;

Separa as informações do formulário em categorias de campos. A tag `<legend></legend>` define um título para as categorias.

```
<fieldset>
  <legend>Dados Pessoais:</legend>
  Nome: <input type="text" name="nome">
  E-mail: <input type="text" name="email">
</fieldset>
```

- Tag `<label></label>`;

A tag **label** deve envolver o texto que faz referência ao campo de formulário. Por exemplo, a palavra “Nome” que precede o campo de preenchimento do nome do usuário. Como atributo da tag **label** utilizamos o *for*, e seu valor deverá ser uma palavra qualquer. A seguir, devemos definir um atributo **id** no campo de formulário com o mesmo valor dado para o atributo *for* da tag **label**.

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

7.3 Campos menos utilizados

- File;

Valor para que seja possível anexar arquivos no envio do formulário. Ao utilizar este valor no formulário, devemos mudar o **enctype** da tag `form` para **multipart/form-data**.

```
<input name="anexar" type="file" size="50" title="Procurar arquivo...">
```

- Password;

Serve para criar um campo de senha, onde os caracteres digitados são ocultos. Os atributos são semelhantes ao valor **text**.

```
<input type="password" name="senha">
```

- Hidden;

Cria um campo oculto. Só faz sentido criá-lo com o uso de *scripts* para funcionalidades.

```
<input type="hidden" name="senha">
```

- Number;

Para criarmos um campo numérico poderemos utilizar o atributo **number** para a tag **input**. Este campo também é útil para dispositivos móveis, trazendo o teclado número nos celulares para preenchimento do campo. Podemos também definir um valor de incremento, máximo e mínimo para o campo:

```
Quantidade: <input type="number" name="quantidade" min="1" max="10" step="0.5">
```

- Range;

O atributo **range** para a tag **input** cria um controle deslizante para uma escolha numérica.

```
Quantidade: <input type="range" name="quantidade" min="10" max="50" value="20" step="1">
```

- Email;

Verifica se no campo inserido foi digitado um valor que corresponda a um e-mail, ou seja, verifica se foi digitado ao menos a arroba, sem espaços e acentos. Caso o usuário não digite um valor semelhante a um endereço de e-mail, o navegador apresentará uma mensagem padrão informando a necessidade do preenchimento.

```
Nome: <input name="e-mail" type="email">
```

- Datalist;

Campo semelhante ao comportamento da *tag select*, mas oferecendo uma lista apenas como sugestões e pesquisa para preenchimento.

```
<label for="browser">Escolha seu navegador:</label>
<input list="browsers" name="browser" id="browser">

<datalist id="browsers">
  <option value="Edge">
  <option value="Firefox">
  <option value="Chrome">
  <option value="Opera">
  <option value="Safari">
</datalist>
```

Obs.: Existe ainda uma gama imensa de tipos de input e seus comportamentos. Verifique a lista completa aqui: [W3Schools - HTML form input Types](https://www.w3schools.com/html/html_form_input_types.asp)

7.4 Botões

Existem vários tipos de botões diferentes para formulários, sempre executando ações específicas ao serem clicados. Sua estilização será feita via **CSS**. Vejamos.

- Submit;

Cria um botão que executa o que foi definido no atributo **action** da *tag form*. O atributo **value** define o texto dentro do botão, se ele for omitido, o navegador atribui um texto padrão.

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

- Image;

Utilizado para inserir uma imagem de um botão ao invés do botão padrão **HTML**. Ele assumirá a função de um botão *submit*, executando o **action** da *tag form*.

```
<input type="image" src="imagens/imagem.jpg">
```

- Reset;

Cria um botão para apagar todos os dados digitados no formulário. Não é uma boa usabilidade utilizá-lo.

```
<input type="reset" value="Limpar campos">
```

- Button;

Cria um botão genérico. Geralmente utilizado para ganhar funcionalidades por eventos **JavaScript** ou links.

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

- Tag **<button></button>**.

Utilizado para criar botões genéricos quando não inseridos dentro de uma *tag form*.

```
<button>Texto do botão</button>
```

Veja como ficaria um formulário utilizando estes campos mais conhecidos:

```
<body>
  <form action="aula.php" method="post">
    <fieldset>
      <legend>Dados pessoais:</legend>
      <ul>
        <li>
          <label for="nome">Nome:</label>
          <input type="text" name="nome" id="nome">
        </li>
        <li>
          <label for="email">E-mail:</label>
          <input type="email" name="email" id="email">
        </li>
        <li>
          Gênero:
          <input type="radio" name="sexo" value="masculino"
id="masculino"><label for="masculino">Masculino</label>
          <input type="radio" name="sexo" value="feminino"
id="feminino"><label for="feminino">Feminino</label>
        </li>
      </ul>
    </fieldset>
    <fieldset>
      <legend>Cursos:</legend>
      <ul>
        <li>
          <input type="checkbox" name="cursos" value="HTML"
id="HTML"><label for="HTML">HTML</label>
          <input type="checkbox" name="cursos" value="CSS"
id="CSS"><label for="CSS">CSS</label>
```

```

        <input type="checkbox" name="cursos" value="JS"
id="JS"><label for="JS">JS</label>
        <input type="checkbox" name="cursos"
value="jQuery" id="jquery"><label for="jquery">jQuery</label>
    </li>
</ul>
</fieldset>
<p>
    <label for="mensagem">Mensagem:</label>
</p>
<p>
    <textarea name="mensagem" id="mensagem"></textarea>
</p>
<p>
    <input type="submit" value="Enviar">
</p>
</form>
</body>

```

7.5 Campos Interativos

Foram criadas novas funções para facilitar o uso dos formulários no **HTML5**, e ajudar o usuário sem a necessidade de *scripts* para isso. Vejamos:

7.5.1 Autocomplete

Os navegadores por padrão guardam os dados digitados em formulários anteriormente, e quando o usuário dá foco ao campo o navegador lhe traz sugestões baseado em seu histórico. Com **HTML5** poderemos ativar ou desativar este recurso com o atributo **autocomplete**. Ele possuirá dois valores, *on* e *off*, sendo *on* o padrão e não havendo necessidade de definir. Este atributo se aplica as *tags* **form** e **input**.

```
<input type="text" name="nome" autocomplete="off">
```

7.5.2 Placeholder

Oferece uma dica sobre o que deve ser preenchido dentro do campo. Pode ser inserido nas *tags* **input** e **textarea**:

```
Nome: <input type="text" name="nome" placeholder="Digite seu
nome completo">
```

7.5.3 Autofocus

Define o objeto que já entrará em foco assim que a página for carregada. Trata-se de um atributo *booleano*. Aplica-se as *tags* **button**, **input**, **select** e **textarea**:

```
Nome: <input type="text" name="nome" autofocus>
```

7.5.4 Validação de formulário

Validar um formulário é torná-lo mais seguro sobre o conteúdo que será inserido pelo usuário e melhora a usabilidade. Esta validação até os dias de hoje sempre foi feita através de **JavaScript**, mas agora poderemos realizar a validação também através do atributo *required*. Se utilizado torna o campo de preenchimento obrigatório. Trata-se de um atributo *booleano*, ou seja, não precisa de valor. Pode ser inserido nas *tags* **input**, **select** e **textarea**:

```
Nome: <input type="text" name="nome" required>
```

Veja o exemplo de um formulário completo utilizando os novos atributos **HTML5**. Salve como **contato.html** para que futuramente possamos estilizá-lo e inseri-lo em uma página:

```
<body>
  <form action="form.php" method="post">
    <fieldset>
      <legend>Dados pessoais:</legend>
      <ul>
        <li>
          <label for="nome">Nome:</label>
          <input type="text" name="nome" id="nome"
placeholder="Digite seu nome completo">
        </li>
        <li>
          <label for="email">E-mail:</label>
          <input type="email" name="email" id="email"
placeholder="exemplo: contato@empresa.com.br" required>
        </li>
        <li>
          <label for="estado">Estado:</label>
          <select name="estado" id="estado">
            <option value="Não definido" selected>Escolha um
Estado</option>
            <option value="Paraná">PR</option>
            <option value="Santa Catarina">SC</option>
```

```

        <option value="Rio Grande do Sul">RS</option>
    </select>
</li>
</ul>
</fieldset>
<fieldset>
    <legend>Produtos:</legend>
    <ul>
        <li>
            <input type="checkbox" name="produtos" id="camiseta"
value="Camiseta"><label for="camiseta">Camiseta </label>
            <label for="qtde_camiseta">Quantidade</label>
            <input type="number" name="qtde_camiseta"
id="qtde_camiseta" min="1" max="10" value="1">
        </li>
        <li>
            <input type="checkbox" name="produtos" id="bone"
value="Boné"><label for="bone">Boné</label>
            <label for="qtde_bone">Quantidade</label>
            <input type="number" name="qtde_bone" id="qtde_bone"
min="1" max="10" value="1">
        </li>
        <li>
            <input type="checkbox" name="produtos" id="adesivo"
value="Adesivo"><label for="adesivo">Adesivo</label>
            <label for="qtde_adesivo">Quantidade</label>
            <input type="number" name="qtde_adesivo" id="qtde_adesivo"
min="1" max="10" value="1">
        </li>
    </ul>
</fieldset>
<p>
    <label for="mensagem">Mensagem:</label>
</p>
<p>
    <textarea name="mensagem" id="mensagem" title="Digite sua
mensagem" placeholder="Digite aqui sua mensagem" required></textarea>
</p>
<p>
    <input type="submit" value="Enviar">
</p>
</form>
</body>

```



CAPÍTULO 8

INTRODUÇÃO ÀS **CSS**

Capítulo 8 - Entendendo as CSS

CSS é a sigla para **Cascading Style Sheets** ou **Folhas de Estilo em Cascata** e servem para a estilização de documentos **HTML**. Ela não terá a função de inserir elementos, e sim personalizar e formatar os elementos já inseridos no código **HTML**. Através das folhas de estilo poderemos alterar toda a aparência do site, e manter um padrão para que todo o projeto siga a mesma formatação. Desta forma economizamos tempo e desgaste no momento da criação e atualização das páginas. Existem três formas de inserirmos as folhas de estilo no documento:

- In-line;
- Incorporado;
- Linkado externamente.

O **CSS In-line** é escrito dentro da *tag*, dando a ela uma estilização válida somente para aquele elemento. Para isso, utilizamos o atributo global **style**. Veja o exemplo:

```
<p style="color: #900; font-weight: bold;">Parágrafo estilizado por CSS in-line</p>
```

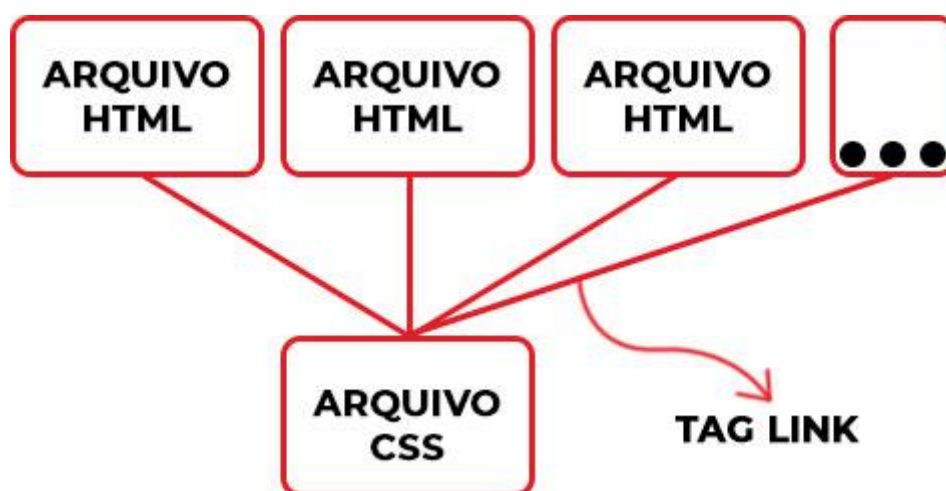
No exemplo, o texto receberia a formatação de negrito e com a cor de letra vermelha. Já o **CSS Incorporado** é escrito todo na seção **head** do documento, e ele especifica a estilização para toda aquela página. Essa formatação é feita através da *tag* **<style></style>**. Veja o exemplo:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8">
    <title>Formulário de contato</title>
    <style>
      p {
        color: #900;
        font-weight: bold;
      }
    </style>
  </head>
  <body>
    <p>Parágrafo estilizado por CSS incorporado</p>
  </body>
</html>
```

Podemos observar que temos a identificação do elemento a ser atingido através da **CSS**, no caso a **tag p**. Na sequência, entre chaves, temos a declaração das propriedades e valores a serem configurados para esta **tag**.

Dica: No **HTML5** podemos inserir a tag `<style></style>` para estilização em qualquer região do documento, não apenas na seção **head**.

No caso do **CSS externo** ou **linkado** criaremos uma única folha de estilo que, ao ser ligada a diversas páginas, terá o poder de estilizar todas elas ao mesmo tempo, pois todas estarão dependendo dela para sua formatação. A imagem a seguir ilustra a ideia:



Este arquivo **CSS** linkado terá a extensão **.css**. Faremos esta ligação das páginas **HTML** com o arquivo **CSS** de estilização através da **tag <link>** vista anteriormente, na seção **head** do documento, e ela deve ser inserida em cada um dos documentos **HTML** em que se deseja a estilização por meio desta folha de estilos. Veja o exemplo:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8">
    <title>Formulário de contato</title>
    <link rel="stylesheet" href="css/style.css">
  </head>
  <body>
    <p>Parágrafo estilizado por CSS incorporado</p>
  </body>
</html>
```

O atributo **rel** mostra a relação da **tag link**, definido como folhas de estilo e o atributo **href** indica o caminho e o nome do arquivo que contém as estilizações (no

exemplo, o nome do arquivo **style.css** contido dentro da pasta **css**). O atributo **media** diz em qual tipo de dispositivo esta folha de estilos deve ser utilizada, o assunto será mais abordado no capítulo sobre **layouts responsivos**.

Fazer uma chamada em cada arquivo **HTML** a um único arquivo **CSS** é a melhor prática, pois diminui o número de requisições do navegador ao servidor, melhorando a performance, e facilita a manutenção do projeto.

8.1 Sintaxe CSS

Observe que as **CSS** são escritas de forma diferente que o **HTML**, com a presença de **dois-pontos**, **chaves** e **ponto-e-vírgula**. Chamamos o elemento que queremos atingir no código **HTML** de **seletor**, o conteúdo entre as chaves de **declaração**, as definições do que será feito de **propriedades** e as especificações das propriedades de **valores**, desta forma:

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

Os espaçamentos são apenas por questão de **indentação**. Perceba que ao final de cada propriedade da declaração temos um dois-pontos, e ao fim de cada valor, um ponto-e-vírgula para mostrar que concluímos a formatação daquela propriedade. Tudo que estiver entre chaves (declaração) atingirá o elemento definido no seletor. Poderemos definir quantas propriedades julgarmos necessárias para cada seletor. Veja um exemplo de um arquivo **CSS linkado** estilizando uma página **HTML**:

```
body {  
    margin: 0;  
    background-color: #CCC;  
    color: #FFF;  
}  
h1 {  
    font-family: "Trebuchet MS", Arial, Helvetica, sans-serif;  
    color: #900;  
    font-size: 26px;  
}  
h2 {  
    font-weight: normal;  
    font-size: 22px;  
    color: #393;  
}  
p {  
    font-size: 16px;  
}
```

No exemplo, foram atingidas as *tags* **body**, **h1**, **h2** e **p** do código **HTML**. Receberam formatações como cor de fundo da página, margens e formatações de texto. Como é um arquivo linkado, se vários arquivos **HTML** se conectarem a esta mesma folha de estilos, todos eles receberão a mesma formatação padrão definida na **CSS**. Atualizar a aparência do site todo também ficará muito mais fácil, podendo mudar a cor de fundo de todas as páginas, por exemplo, alterando apenas uma única linha de código.

IMPORTANTE: Note que no valor 0 (zero) para as margens não foi definido uma unidade de medida, já nas demais foi definido px como padrão para pixels. Nas **CSS** sempre precisaremos definir a unidade de medida ao qual os números se referem, que podem ser **px** (pixels), **%** (porcentagem), **cm** (centímetro), **pt** (ponto) ou **in** (polegada), a menos que o valor seja 0 (zero).

8.2 Comentários

Fazer um comentário por **CSS** é muito simples. Ele será iniciado por uma barra seguida de um asterisco, terminando de forma invertida:

```
/* Corpo do site */  
body {  
    margin: 3px; /* Definição das margens do documento */  
}
```

Os comentários podem ser inseridos em qualquer lugar do código **CSS**, desde que não atrapalhem o desenvolvimento do código, e são muito úteis para nos ajudar a entender o conteúdo, principalmente para códigos extensos. O navegador sempre ignora os comentários.

8.3 Efeito Cascata

As folhas de estilo são chamadas de cascata devido a uma hierarquia existente entre elas. Digamos que temos um documento conflitante, onde em uma **CSS in-line** pedimos para determinada *tag* **HTML** tenha seu texto estilizado em vermelho, uma **CSS incorporada** pedindo para que a mesma *tag* tenha seu texto estilizado em azul, e uma **CSS linkada** pedindo para que a mesma tenha o seu texto estilizado em verde. Qual prevalecerá? Para resolver esta questão foi criada uma hierarquia, que envolverá vários pontos.

8.3.1 Hierarquia de tipo de arquivo

O primeiro ponto a pesar na hierarquia de efeito cascata será sempre de dentro para fora, ou seja, a estilização que está mais perto da *tag* (**in-line**) prevalecerá sobre as outras e assim sucessivamente. Caso não haja **CSS in-line**, prevalecerá a **incorporada**, caso não haja nenhuma outra, predomina a **CSS linkada**. Essa hierarquia é útil para momentos em que gostaríamos que todo o site tivesse uma determinada estilização, mas com alguma exceção, neste caso inserimos uma **CSS** mais local para o elemento específico.

Obs.: Se algum valor **CSS** for declarado com a definição **!important** ele terá prioridade sobre os demais, ignorando o efeito cascata.

8.3.2 Hierarquia por Seletor

Imaginemos um cenário onde você realizou todos os passos dentro de todas as boas práticas de uso das **CSS**, ou seja, você linkou todos os arquivos **HTML** do seu projeto em uma única folha de estilos. Geralmente este arquivo fica bem extenso, dando estilos para diversos elementos do projeto, e pode acontecer diversas vezes de neste mesmo arquivo elementos estarem sendo estilizados mais de uma vez, ou seja, mais uma vez tivemos um empate, e o efeito cascata precisa entrar em ação. Em casos assim, o critério do efeito cascata se chama especificidade, ou seja, o seletor que for mais específico terá mais peso, e seu estilo prevalecerá sobre outros.

Em paralelo com a especificidade existe um peso de seletores, que pode ser visualizado com alguns exemplos nesta tabela:

Seletor	Id (centena)	Class (dezena)	Elemento (tag)	TOTAL
p	0	0	1	1
.texto	0	1	0	10
#texto	1	0	0	100
.texto p	0	1	1	11
#texto p	1	0	1	101
.texto ul li	0	1	2	12

Quanto maior a soma, mais forte o seletor. Como é possível observar, o seletor de elemento é o mais fraco, sendo rapidamente sobreposto por um seletor que ataca o

mesmo elemento, mas pela classe, e este por sua vez é mais fraco do que um seletor por Id.

8.3.3 Hierarquia por Posição

No caso de todas as lógicas anteriores de hierarquia do efeito cascata terem falhado, a posição do seletor dentro do arquivo **CSS** é o último critério. Basicamente, o elemento que vier por último, ou seja, que estiver mais abaixo no arquivo da folha de estilos será o mais forte.

8.4 Herança

Logo mais aprenderemos como desenvolver seletores **CSS** e propriedades, mas por hora já vale a pena aprendermos a definição de herança. Algumas propriedades **CSS**, em especial as de textos, são herdáveis, ou seja, você aplica um estilo à *tag* pai, e todos os elementos descendentes (*tags* dentro de outras *tags*) receberão este mesmo estilo. Veja o exemplo:

```
nav {  
  color: #900;  
}  
nav ul li {  
  color: #CCC;  
}
```

Neste caso, somente receberão a primeira cor aqueles elementos dentro da *tag* **nav** não contidos em listas, pois os contidos em listas receberão a segunda cor. Se não tivéssemos definido uma formatação específica para listas, todos os textos contidos em *tag* **nav** também receberiam a mesma formatação por **herança**.



CAPÍTULO 9

SELETORES **CSS**

Capítulo 9 - Seletores

Como vimos na sintaxe **CSS**, os seletores são a primeira etapa para se escrever uma regra **CSS**, é nele que diremos qual parte da página **HTML** gostaríamos de estilizar. Um bom desenvolvedor precisa de um domínio em seletores, vejamos alguns:

9.1 Seletor universal

Existe um elemento chave, que atingirá todas as *tags* do documento, representado por um asterisco. Geralmente usa-se este elemento para remover já de antemão bordas, margens e espaçamentos, veja seu uso:

```
* {  
  margin: 0;  
  padding: 0;  
  border: 0;  
  vertical-align: baseline;  
}
```

No exemplo foram zerados valores como margens e espaçamentos. Isso é chamado pelos profissionais de *reset*. Existem resets considerados melhores do que este, já que este atinge realmente todas as *tags*, tornando a página mais lenta. Um bom *reset* conhecido é o do **Eric Meyer**, que pode ser encontrado em <http://meyerweb.com/eric/tools/css/reset>.

9.2 Elemento

O seletor mais comum do **CSS** é o de elemento. É quando gostaríamos de atingir uma tag específica do **HTML** em todas as suas ocorrências. É um seletor muito genérico, então deve ser utilizado com cuidado, como em *resets*.

```
p {  
  color: #900;  
}
```

9.3 Agrupamentos

Podem haver casos em que gostaríamos que várias *tags* **HTML** recebessem a mesma formatação **CSS**, por exemplo, que todos os cabeçalhos da página tenham o

mesmo tipo de letra, variando somente em cores e tamanhos. Para isto, definimos todas as *tags* que gostaríamos de atingir no mesmo seletor, separando por vírgula:

```
h1, h2, h3, h4, h5, h6 {  
  font-family: "Trebuchet MS", Arial, Helvetica, sans-serif;  
}
```

9.4 Classe e Id

Muitas vezes precisaremos identificar no código **HTML** alguns elementos para assim podermos atingi-los pelas **CSS**. Identificamos os elementos de duas formas diferentes:

- **Id**;

Podemos dar o atributo **id** para qualquer *tag HTML*, por ser um atributo global. O valor do atributo deve ser uma identificação única no arquivo, não podendo haver no mesmo arquivo duas **id's** com o mesmo nome. Para evitar contratempos, utilize caracteres de A-Z, ou traços e *underline*. Exemplo:

```
<h1 id="texto">Texto formatado</h1>
```

Uma vez identificado o elemento, iremos atingi-lo via **CSS** através de uma cerquilha seguida da identificação dada ao elemento no atributo **id**:

```
#texto {  
  color: #900;  
  font-weight: normal;  
}
```

As **id's** são utilizadas quando queremos estilizar um elemento único do código **HTML**.

- **Class**

Agora imagine que precisamos estilizar uma galeria de 50 imagens do site, colocando bordas para todas elas. Não poderíamos atingir a *tag* **** sob risco de estilizar imagens que não gostaríamos de estilizar também, como uma logo. Também não seria correto utilizar a mesma **id** para todas elas, pois um valor de **id** não pode se repetir na mesma página. Então neste caso utilizaremos **class**. Veja, usa-se **id** em uma *tag HTML*

para atingir um único elemento, e **class** para um grupo de elementos semelhantes. Observe como ficaria o código **HTML** seguida da estilização das **classes** por **CSS**:

```





...
```

Atribuindo a mesma **class** a todos os elementos **HTML**, atingiremos a todos eles pelas **CSS** ao mesmo tempo. Fazemos isto através de um ponto seguido do nome da **class** que gostaríamos de estilizar:

```
.galeria {
  border: 2px ridge #900;
}
```

Pronto, todas as imagens terão a mesma formatação, e o melhor, caso queiramos alterar alguma formatação para a galeria toda, como a cor da borda por exemplo, faremos isto alterando apenas uma única linha de código. **Class** também é um atributo global e pode ser inserida em qualquer **tag HTML**, mas esta pode ser repetida quantas vezes desejar.

Caso queiramos que determinada **class** funcione somente quando se refere a determinada **tag**, por exemplo um texto sendo formatado por uma **class**, mas somente quando aparecer dentro da **tag p**. Supondo que no código **HTML** temos a seguinte formação: **<p class="texto"> Texto exemplo </p>**, o código **CSS** ficaria assim:

```
p.texto {
  font-family: Arial, Helvetica, sans-serif;
  font-size: 16px;
}
```

Caso haja a mesma **class** em outras **tags** ela não receberá a formatação, somente quando ocorrer em **tag p**.

9.5 Elemento descendente

Outro momento é quando temos a definição de elementos descendentes, ou seja, uma **tag** está inserida dentro de outra. Se quisermos atingir os elementos só quando dentro de outra **tag**, separaremos os elementos declarados no seletor por um espaço em branco:

```
.box li a{  
  color: #900;  
}
```

No exemplo, pedimos que todo o *link* que estiver em uma lista, e que essa lista esteja dentro de um elemento com a **class** box, tenha a cor de letra vermelha.

9.6 Pseudo-classes

Pseudo-classes são elementos estilizados por CSS, mas em uma ocasião específica. Por exemplo, podemos atingir um elemento apenas quando o usuário passar o mouse sobre ele. Em sua sintaxe define-se o elemento a ser atingido, seguido de dois-pontos, e em seguida o **pseudo-classes** que gostaríamos de aplicar. Vejamos alguns exemplos.

- hover;

Momento em que o usuário posiciona o mouse sobre o elemento.

```
a:hover{  
  color: #03C;  
  text-decoration: underline;  
}
```

- focus;

Momento em que um elemento, como um campo de formulário, entra em foco.

```
input:focus{  
  border:1px solid #900;  
}
```

- last-child;

Seleciona o último elemento de uma área, como o último item de uma lista.

```
.box li:last-child{  
  border:1px solid #900;  
}
```

- first-child;

Seleciona o primeiro elemento de uma área, como o último item de uma lista.

```
.box li:first-child{  
  border:1px solid #900;  
}
```

- nth-child(n).

Seleciona um elemento descendente específico.

```
.box li:nth-child(3){  
  border:1px solid #900;  
}
```

Neste exemplo, o terceiro item de lista seria selecionado para ser estilizado. A pseudo-classe nth-child também aceita valores odd e even (par ou ímpar).

9.7 Pseudo-elementos

Pseudo-elementos permitem a criação de elementos **HTML** para sua estilização. Utilizamos duas vezes dois-pontos após o elemento a ser estilizado, seguido o **pseudo-elemento**.

- first-letter;

Para a estilização da primeira letra de um elemento **HTML**.

```
p::first-letter{  
  font-size: 18px;  
}
```

- first-line;

Para a estilização da primeira linha de um texto.

```
p::first-line{  
  font-size: 18px;  
}
```

- after;

Permite a inserção de conteúdo após um elemento. Deve-se declarar a propriedade **content** para definir o conteúdo a ser inserido.

```
p::after{  
  content:"Texto de exemplo";  
}
```

```
display:block;
font-size:18px;
}
```

- before.

Semelhante ao **after**, mas permitindo a inserção de conteúdo antes de um elemento.

```
p::before{
  content:"Texto de exemplo";
  display:block;
  font-size:18px;
}
```

9.8 Seletores avançados

9.8.1 Seletor de atributo

Podemos seleccionar uma *tag* no documento **HTML**, mas só quando essa possuir determinado atributo, como no exemplo:

```
input[type="submit"] {
  background: #900;
}
```

No exemplo foi seleccionada uma *tag* **input**, mas só quando esta possuir o atributo **type="submit"**. Para fazer a separação entre *tag* e atributo, utiliza-se o sinal de colchetes.

9.8.2 Elemento filho

Caso não queiramos que os elementos filhos dos filhos recebam um valor, podemos definir um sinal de maior que (>), para que apenas os filhos diretos da *tag* pai recebam essa estilização, como no exemplo:

```
nav > ul > li {
  color: #CCC;
}
```

Neste caso, se houvessem sublistas dentro desta lista, elas não seriam estilizadas, pois o seletor aponta apenas para filhos diretos.

9.8.3 Elemento irmão

Também podemos apontar para elementos irmãos, ou seja, quando as tags possuem o mesmo pai direto, como no exemplo:

```
p ~ p {  
  color: #CCC;  
}
```

Utiliza-se o acento “til” para apontar para uma *tag* irmã. No exemplo, seria estilizada uma *tag* **p**, mas só quando ela se entrar ao lado de outra *tag* **p**.



CAPÍTULO 10

PROPRIEDADES **CSS**

Capítulo 10 - Propriedades CSS

As propriedades CSS podem ser atribuídas a qualquer seletor. Confira uma lista completa no final da apostila de todas as possibilidades CSS e das novas propriedades do CSS3. Vamos aprender as mais utilizadas em um projeto web.

10.1 Valores para propriedades

Toda propriedade CSS possuirá um valor. Estes valores podem ter algumas especificações que demandam conhecermos alguns detalhes, vamos conhecer:

10.1.1 Unidades de medida

Muitas propriedades precisam ser declaradas com um valor numérico, mas no CSS devemos sempre especificar a unidade de medida referente àquele valor numérico (salvo casos do valor ser zero). As unidades podem ser divididas em absolutas e relativas, vejamos algumas dessas unidades:

Relativa (com base em uma referência)	Absoluta (Global)
px (pixels). É considerada relativa pois depende do pixel físico do dispositivo, mas se comporta como absoluta.	pt (pontos).
% (porcentagem). Depende da referência do elemento pai.	in (polegadas).
em. Depende da referência do elemento pai.	mm. (milímetros).
rem. Depende da referência da tag body.	pc. (paicas)

Unidades de medida absolutas são menos utilizadas. A unidade px é comumente utilizada para estabelecer referências. As unidades %, em e rem são muito utilizadas em layouts responsivos, que veremos mais adiante.

10.1.2 Cores

Diversas propriedades CSS esperam como seu valor uma cor, e existem diversas formas de se declarar cores no CSS:

Nome em inglês	Hexadecimal	rgb	rgba
black	#000	rgb(0,0,0)	rgba(0,0,0,1)
white	#fff	rgb(255,255,255)	rgba(255,255,255,1)
gray	#ccc	rgb(204,204,204)	rgba(204,204,204,1)
purple	#800080	rgb(128,0,128)	rgba(128,0,128,1)

O nome em inglês possui diversas variações, mas é pouco utilizado. O código hexadecimal é bastante utilizado, e percebe-se que ele pode ser escrito só com 3 letras e números ou com 6. Isso ocorre porque quando a cor possui três pares perfeitos, como em #9900CC, ela pode ser encurtada para #90C, mas somente quando forem duplas perfeitas, no caso do código #9900C1, ele não pode ser encurtado.

O código rgb é semelhante ao hexadecimal mas escrito de outra forma. Já o rgba é bastante utilizado quando precisamos colocar transparência na cor, visto que o último valor, chamado de *alpha*, diz respeito a esta transparência, e pode receber valores de zero à 1.

10.1.3 Fontes

Em momentos bem específicos precisaremos declarar fontes como valor das propriedades. Elas sempre devem ser declaradas em família, separadas por vírgulas, pois caso o usuário não possua determinada fonte em seu computador, outra deverá substituí-la. No capítulo de **CSS3** nos aprofundaremos no assunto.

10.2 Formatação de textos

As propriedades CSS para formatação de textos são usualmente aplicadas a seletores das *tags* de cabeçalho, de parágrafo, ou com **class** e **id** específicas. Veja uma tabela prática com as principais propriedades para formatação de textos:

Propriedade	Função	Valores
Font-family	Alterar o tipo da letra.	Família de fontes utilizada.
Font-size	Tamanho da letra.	Valor numérico.
Font-style	Define valores como Itálico.	Italic.
Font-weight	Define valores como Negrito.	Bold, normal. Também aceita valores de 100 a 900.

Color	Cor da letra.	Código hexadecimal da cor, rgb, rgba ou nome da cor em inglês.
Text-align	Alinhamento de texto.	Center, justify, left ou right.
Text-decoration	Sublinhado.	Underline, overline, line-through ou none.
Text-transform	Altera a caixa do texto.	Uppercase, lowercase e capitalize.
line-height	Altera a altura da linha.	Valor numérico. Pode ser definida juntamente com a propriedade font-size.
word-spacing	Espaçamento entre palavras.	Valor numérico.
letter-spacing	Espaçamento entre letras.	Valor numérico.

Exemplo:

```
h1 {
  font-family: Verdana, Geneva, sans-serif;
  font-size: 14px;
  color: #630;
  text-align: center;
  font-weight: normal;
  font-style: italic;
  text-decoration: underline;
}
```

É possível declarar todas os valores da propriedade **font** de forma agrupada, desde que sempre haja a declaração de tamanho e tipo de letra e com as estilizações antes do mesmo, desta forma:

```
h1 {
  font: normal 14px Verdana, Geneva, sans-serif;
}
```

Veja, foi declarado um valor para remover o negrito, um para tamanho de letra, e outro para tipo de letra, necessariamente nesta ordem.

10.3 Listas

Existem propriedades **CSS** para personalizarmos as listas ordenadas e não ordenadas.

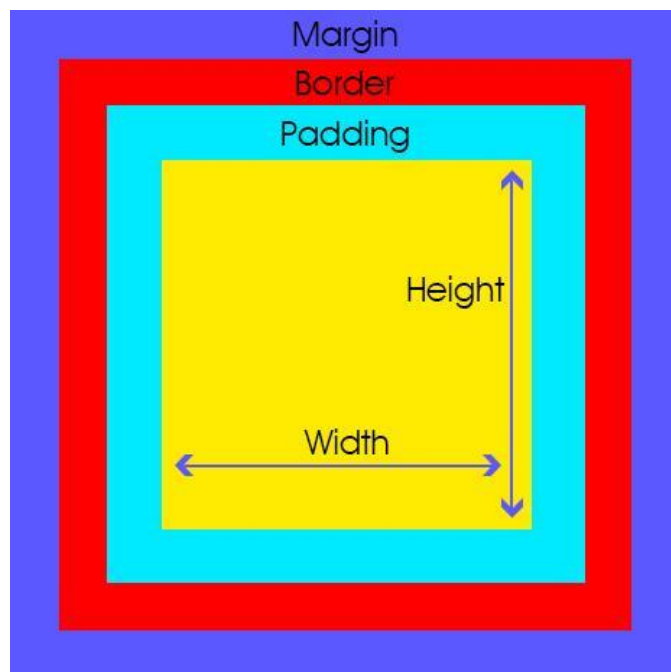
Propriedade	Função	Valores
List-style-type	Define o tipo de marcador.	Para uma lista com marcadores personalizados poderemos utilizar circle (círculo), square (quadrado), disc (círculo preenchido), entre outros.
List-style-position	Define a posição do marcador.	inside ou outside.
List-style-image	Define uma imagem como marcador.	url(“imagem.jpg”).

É possível agrupar em apenas uma declaração, desta forma:

```
li{
  list-style:inside square;
}
```

10.4 Propriedades para o Box-model

Os elementos do **HTML** tem uma forma padrão de serem exibidos, alguns deles são elementos chamados de elementos em linha (*inline*) ou em bloco (*block*). Os elementos em bloco podem ser estilizados tomando como base um box-model, que seria um modelo de como as propriedades **CSS** se aplicam a ele. Veja o exemplo:



Margem é o espaçamento dado à área externa do *box* onde é inserida a propriedade. Atingiremos as margens do seletor com a definição da propriedade **margin**. *Padding* é o nome dado para o espaçamento interno do elemento. Ele funciona como um enchimento, dando o espaçamento interno, mas fazendo o *box* crescer de forma diretamente proporcional ao valor atribuído à propriedade **padding**. Também é possível atribuir a propriedade **border** ao *box*, e essa borda será posicionada entre a margem e o padding. **Border** também aumentará o tamanho do *box* de forma diretamente proporcional. **Width** e **height** são as propriedades para alterar o tamanho do *box*, em largura e altura respectivamente. Vejamos como trabalhar com as propriedades relacionadas ao *Box-model*.

10.4.1 Display

Existem elementos que por característica são elementos de linha, e outros elementos de bloco. Por exemplo, as tags **span**, **strong** e **a** são elementos de linha, já as tags **div**, **h1** e **p** são elementos de bloco, ou ainda temos casos de exibição por dados tabulares, como no caso de tabelas, pela forma em que elas exibem as informações. A propriedade **display** servirá para transformar essa forma de exibição. Por exemplo, podemos fazer com que uma lista ordenada seja exibida em linha, ou um *link* em bloco. Veja os valores para a propriedade **display**:

Valores	Função
Inline	Muda a exibição do elemento para em linha.
Block	Muda a exibição do elemento para em bloco.
None	Não faz alteração na exibição do elemento ou retorna para o padrão.
Table	Muda a exibição do elemento para dados tabulares.
Inline-block	Muda a exibição do elemento para um elemento que se comporta ao mesmo tempo como linha e como bloco.

Exemplo:

```
a{
  display: block;
}
```

IMPORTANTE: Apenas elementos de blocos receberão propriedades de *box-model*. Ou seja, se você gostaria que um *link*, por exemplo, recebesse propriedades de bloco, deveria mudar seu **display** para **block**.

10.4.2 Margin

No caso de margens temos várias formas de trabalhar. Primeiro, especificando valores para cada uma delas:

```
.box{
  margin-top: 5px;
  margin-right: 3px;
  margin-bottom: 10px;
  margin-left: 3px;
}
```

Note que especificamos valores para cada uma das margens, sendo *top* a margem superior, *right* a margem direita, *bottom* a margem inferior e *left* a margem esquerda. Também poderíamos atingi-las em uma única propriedade:

```
.box{
  margin: 5px 3px 10px 3px;
}
```

A ordem para inserção é sempre a mesma e lembra os ponteiros de um relógio, sendo o primeiro valor sempre se referindo ao *top*, o segundo se referindo ao *right*, o terceiro se referindo ao *bottom* e por último o *left*. Também é possível fazer a margem por três eixos, sendo o primeiro referente ao *top*, o segundo *left* e *right* e o terceiro ao *bottom*, ficando assim:

```
.box{
  margin: 3px 5px 10px;
}
```

Podemos ainda atingir as margens por dois eixos, sendo um primeiro valor para o eixo vertical e o segundo para o eixo horizontal:

```
.box{
  margin: 3px 10px;
}
```

No exemplo, foi atribuído o valor de 3 pixels de margem para as margens superior e inferior e 10 pixels para as margens esquerda e direita.

Dica: Se definirmos `margin: 0 auto` para um elemento de bloco, ele se centralizará horizontalmente onde estiver inserido, pois define zero para o *top* e *bottom*, e o *default* centralizado para as demais.

Quando os valores são os mesmos, a especificação de apenas um valor aplicará esse valor às quatro margens:

```
.box{  
  margin: 3px;  
}
```

No exemplo, todas as margens terão o valor de 3 pixels.

10.4.3 Padding

Outro espaçamento comumente utilizado é o **padding**, que define o espaçamento entre o limite do elemento e o conteúdo nele inserido, como no caso de tabelas e `div`'s. Trabalhar com **padding** é semelhante às margens em suas propriedades e valores. Lembrando que o valor definido para **padding** irá aumentar o tamanho do *box*.

```
div {  
  padding: 20px;  
}
```

Utilizaremos **margin** quando queremos alterar o espaçamento entre os limites do conteúdo e sua área externa, e **padding** quando gostaríamos de alterar o espaçamento entre os limites do conteúdo e sua área interna.

10.4.4 Border

Podemos alterar pelas folhas de estilo as bordas de uma tabela, imagem, `div` ou outros elementos. Para isso, utilizamos essas principais propriedades CSS para **border**:

Propriedade	Função	Valores
Border-width	Largura da borda.	Valor numérico para largura. É comum definir 0 (zero) para links em imagens por questões estéticas.
Border-color	Cor da borda.	Código hexadecimal da cor, rgb, rgba ou nome da cor em inglês.

Border-style	Estilo da borda.	Dashed, dotted, ridge, double, solid são alguns exemplos.
--------------	------------------	---

Exemplo:

```
div {
  border-width: 3px;
  border-color: #900;
  border-style: dashed;
}
```

Ou pode ser escrita em uma única linha:

```
div {
  border: 3px dashed #900;
}
```

Se quisermos também poderemos alterar cada borda de forma única, por exemplo, apenas a borda de baixo, ou apenas a da esquerda e direita, definindo qual borda gostaríamos de alterar com *top* (cima), *right* (direita), *bottom* (baixo) ou *left* (esquerda), ficando da seguinte forma:

```
.container {
  border-top: 3px dashed #900;
  border-right: 2px dotted #CCC;
  border-bottom: 3px dashed #900;
  border-left: 2px dotted #CCC;
}
```

10.4.5 Width e Height

Também temos as propriedades para definir tamanho. São elas **width** e **height**, para largura e altura respectivamente.

Propriedade	Função	Valores
Width	Largura	Valor numérico para largura.
Height	Altura	Valor numérico para altura.
Min-width	Largura mínima	Valor numérico para largura mínima que o elemento poderá ter.

Max-width	Largura máxima	Valor numérico para largura máxima que o elemento poderá ter.
Min-height	Altura mínima	Valor numérico para altura mínima que o elemento poderá ter.
Max-height	Altura máxima	Valor numérico para altura máxima que o elemento poderá ter.

```
div {
  width: 200px;
  height: 300px;
}
```

10.4.6 Background

As propriedades **CSS** para alterar o fundo são utilizadas para alterar o fundo da página ou de qualquer elemento de bloco. Veja uma tabela prática com as principais propriedades para alteração de fundo:

Propriedade	Função	Valores
Background-color	Cor de fundo	Código hexadecimal da cor, rgb, rgba ou nome da cor em inglês.
Background-image	Imagem de fundo	url(caminho/imagem.jpg).
Background-repeat	Tipo de repetição da imagem de fundo	No-repeat, repeat-x ou repeat-y.
Background-position	Posição da imagem de fundo	Valor numérico.
Background-attachment	Rolagem do fundo	Fixed.

Exemplo:

```
body {
  background-color: #CCC;
  background-image: url(../imagens/fundo.png);
  background-repeat: no-repeat;
  background-position: 400px 100px;
  background-attachment: fixed;
}
```

Também poderemos escrever todos os valores de forma resumida, em qualquer ordem:

```
body {  
  background: url(../imagens/fundo.png) no-repeat 400px 100px fixed  
#CCC;  
}
```

Neste exemplo vemos as mesmas propriedades do anterior, mas com menor necessidade de código para isso. Caso não vá utilizar todos os valores para a propriedade, basta pular para o próximo valor.

10.4.7 Overflow

A propriedade **overflow** trabalha com elementos que vão além dos limites do *box-model*. Por exemplo, se definirmos um tamanho para um *box*, mas o conteúdo for maior, todo o conteúdo ainda será exibido, e também o tamanho original do *box* será mantido, e os elementos que foram parar além dos limites do *box* são trabalhados com a propriedade **overflow**.

Propriedade	Função	Valores
Overflow	Atribuir um comportamento para elementos além dos limites do <i>box</i> .	Auto, hidden e scroll.

Exemplo:

```
textarea {  
  overflow:hidden;  
}
```

10.4.8 Cursor

Através da propriedade **cursor** poderemos personalizar o cursor do mouse do usuário, como se tornará ao passar sobre o elemento onde a propriedade está inserida. Possui diversos valores, mas o mais comumente utilizado é o **pointer**, para que o ponteiro do mouse se torne em uma mão indicadora, muito interessante para o caso de botões de formulários. Veja o exemplo:

```
input {  
  cursor:pointer;
```

}

10.4.9 Box-sizing

Uma propriedade **CSS** para aqueles que gostariam que **padding** e **border** não alterem o tamanho do *box*. Para isso, basta declarar como no exemplo:

```
.box {  
  width: 300px;  
  height: 100px;  
  border: 1px solid blue;  
  box-sizing: border-box;  
}
```

10.5 Tableless

Nos anos 90 utilizam-se tabelas para estruturação de *sites*. Como a prática não é semântica, em 2001 com o advento do **CSS2** vieram-se técnicas para estruturação de *layout* sem tabelas, essa técnica é chamada de **Tableless**. Faremos isso através de *tags* para *box*. Div's são elementos inseridos no documento para criar divisões onde podemos inserir qualquer conteúdo. É um elemento isento de semântica, pois pode ser aplicado para qualquer finalidade. A div é inserida através da *tag* `<div></div>`. Como vimos em capítulos anteriores, poderemos estruturar todo o *layout* de uma página através das div's, mas como hoje podemos fazer esta função por **HTML5**, ela é mais utilizada para blocos de conteúdo específicos no *layout* e não tanto para estrutura. As div's sem **CSS** não são exibidas, nem tem alguma utilidade em si além de separar os elementos, são as **CSS** que irão desenvolvê-las. O que se faz então é dar sempre uma **id** ou **class** para elas, e então configuramos as mesmas nas folhas de estilo. As propriedades **CSS** atribuídas as div's são aquelas que já aprendemos, como estilizar seu tamanho e estilos. As novidades serão na forma de posicionar esses elementos, vejamos algumas técnicas:

10.5.1 Position

Primeiramente, defina as propriedades do *box-model* para poder visualizar o *box*. Por padrão, esse *box* sempre empurrará os demais elementos para baixo. Através de posicionamento **position**, as div's poderão ser sobrepostas e movimentadas através das propriedades específicas para isso. Vejamos:

Propriedade	Função	Valores
Position	Define a referência para o posicionamento.	Absolute, Relative, Fixed ou Sticky.
Z-index	Define a ordem do posicionamento das div's.	Valor numérico. Quanto maior o valor, mais à frente a div ficará.
Left	Posicionamento em relação à esquerda.	Valor numérico.
Top	Posicionamento em relação ao topo.	Valor numérico.
Right	Posicionamento em relação à direita.	Valor numérico.
Bottom	Posicionamento em relação à base.	Valor numérico.

Position em geral é uma propriedade utilizada para modais (elementos sobrepostos no site), menus *popup* e interações que exijam a sobreposição de objetos, como *mini-carts*.

10.5.2 Float

Float é uma propriedade relativamente simples para permitir que dois elementos de bloco se posicionem um ao lado do outro. No entanto, é preciso ter cuidado e atenção para que o layout não “quebre”, para isso fazendo uso da propriedade **clear** e **overflow**.

Propriedade	Função	Valores
Float	Flutua o elemento na página.	Left, right ou none.
Clear	Faz com que o elemento se comporte respeitando elementos flutuados.	Left, right ou both.

Exemplo:

```
.box{
  width: 35%;
  height: 450px;
  float: right;
}
```

10.5.3 Display Inline-block

Uma técnica muito comum de estruturação de layout é alterar o display dos blocos para **inline-block**. Isso permitirá que os elementos fiquem um ao lado do outro, e ainda continuar recebendo propriedades de bloco. Essa propriedade possibilita o uso de outra propriedade chamada **vertical-align**, que pode receber valores como *middle*, *top*, *bottom* e outros. Exemplo:

```
.box{  
  display:inline-block;  
  vertical-align:top;  
}
```

10.5.4 Display Flex

A técnica mais atual para se fazer tableless é alterar o display do bloco pai para flex. Com apenas esta alteração, todos os elementos filhos se ajustam e se distribuem um ao lado do outro. Veja um exemplo:

```
.box-pai{  
  display:flex;  
}
```

Outra vantagem de se utilizar flex é que a propriedade possui diversas outras propriedades derivadas para que possamos estilizar nosso layout de diversas formas, vejamos as mais utilizadas:

Propriedade	Função	Valores
align-items	Alinha os elementos verticalmente.	flex-start, flex-end, center, space-between, space-around e space-evenly.
justify-content	Alinha os elementos horizontalmente.	Left, right ou both.

Para os elementos filhos podemos atribuir:

Propriedade	Função	Valores
order	Altera a posição do elemento.	Valor numérico.
flex-grow	Aumenta o tamanho do elemento.	Valor numérico.
flex-shrink	Diminui o tamanho do elemento.	Valor numérico.

Uma lista completa de todas as possibilidades com flex pode ser encontrada em [CSS Tricks - A Guide to FlexBox](#).

Você poderá ver esses métodos de estruturação de *layout* de forma aplicada e explicada no site [Learn Layout](#).

Veja um exemplo de um arquivo CSS estilizando um arquivo **HTML**:

```
*{
  margin:0;
  padding:0;
  border:0;
  list-style:none;
}
body{
  font:12px Arial, Helvetica, sans-serif;
  background:#CCC;
}
.wrap{
  margin:0 auto;
  width:1170px;
}
.header{
  background:url(../imagens/fundo.jpg) repeat-x transparent;
  padding:10px 0;
}
.content{
  background-color:#000;
}
.esquerda,
.direita,
.centro{
  float:left;
  width:300px;
  margin:10px;
}
.esquerda{
  background:#900;
}
.centro {
  background:#090;
}
```

```
.direita{
  background:#009;
}
.footer{
  clear:both;
  background:#666;
}
p, h1, h2, h3, h4, a{
  font-family:"Trebuchet MS", Arial, Helvetica, sans-serif;
}
h1{
  font-size:26px;
  text-align:center;
}
a:hover{
  color: #000;
  text-decoration: underline;
}
```



CAPÍTULO 11

TRABALHANDO COM **CSS3**

Capítulo 11 - Trabalhando com CSS3

Temos uma nova versão para folhas de estilo que trazem efeitos e funcionalidades a serem utilizadas nos navegadores mais modernos. Através das **CSS3** poderemos colocar sombras, arredondamentos e até animações. Muitos efeitos funcionarão em todos os navegadores modernos, já alguns apenas para navegadores específicos, outros teremos que criar um código para adaptar o efeito para os demais navegadores. Vejamos a seguir alguns efeitos possíveis de serem feitos por **CSS3**.

11.1 Sombreamento de caixas

Através de **CSS3** podemos fazer efeitos como sombra. São geralmente aplicados a *tags* `div` ou elementos em bloco do **HTML**. Veja:

```
.box{  
  box-shadow: 2px 2px 3px #CCC;  
}
```

O primeiro valor do atributo **box-shadow** define o deslocamento da sombra em relação ao topo, o segundo valor define o deslocamento da sombra em relação à esquerda, o terceiro valor define o enevoamento e o último a cor. É possível definir mais sombras com o uso de vírgulas e repetindo o processo. Também é possível iniciar a declaração do valor com o termo **inset**, para fazermos sombras internas.

11.2 Sombreamento de textos

Um efeito de sombreamento semelhante ao **box-shadow**, mas aplicado a textos é o **text-shadow**. Veja sua aplicação:

```
p.exemplo1 {  
  text-shadow: 2px 2px 3px #CCC;  
}
```

11.3 Cantos arredondados

Um efeito inovador do **CSS3** é a possibilidade de arredondarmos os cantos de `div`'s ou elementos de bloco, através do atributo **border-radius**:

```
#container {
  width: 300px;
  height: 300px;
  border-radius: 3px;
}
```

No exemplo temos a definição de 3 pixels de arredondamento. Podemos também definir diversos valores para cantos diferentes terem arredondamentos diferentes, seguindo uma lógica semelhante à lógica de margin com diversos valores. Para uma criação mais fácil de cantos arredondados personalizados, acesse o site [Border Radius](#).

11.4 Transparência

Outro efeito muito interessante que o **CSS3** nos proporciona é a transparência nos elementos, utilizada principalmente para div's e imagens. O efeito acontecerá através da declaração da propriedade **opacity**:

```
#container {
  width: 300px;
  height: 300px;
  background-color: #CCC;
  opacity: 0.4;
}
```

Simples assim. Teremos uma escala de transparência que varia entre 0 para totalmente transparente e 1 para totalmente sólido, e a variação dos valores entre 0 e 1 para escolha da transparência desejada.

11.5 Filtros

Através da propriedade filter podemos alterar a aparência de imagens e elementos dando efeitos visuais. Veja alguns valores possíveis:

Valores	Descrições
grayscale(100%)	Escala de cinza. O valor 100% faz o elemento ficar preto-e-branco.
blur(10px)	Torna o elemento desfocado.
brightness(50%)	Aumenta o brilho do elemento.
sepia(50%)	Efeito sépia do elemento.

hue-rotate(100deg)	Altera a cor do elemento com base no círculo cromático.
--------------------	---

11.6 Múltiplos Backgrounds

As **CSS3** agora permitem múltiplos backgrounds, podendo atribuir várias imagens de fundo para um mesmo elemento, como no exemplo:

```
div {  
  background: url(../imagens/fundo1.png) no-repeat 400px 100px,  
             url(../imagens/fundo3.png) no-repeat top left #CCC;  
}
```

Note que é declarado uma única propriedade **background**, com vários valores separados por vírgulas. A ordem de posicionamento da imagem será a mesma da declaração dos valores, sendo a primeira imagem de fundo declarada aquela que ficará por cima das outras, e assim por diante. É interessante declarar para a última imagem de fundo também uma cor no final, para as áreas que as imagens de fundo não alcancem.

11.7 Gradiente

Podemos com **CSS3** criar gradientes apenas utilizando códigos. Para isso, definiremos o gradiente desejado na já conhecida propriedade **background-image**:

```
div {  
  background-image: linear-gradient(red, yellow);  
}
```

Você também pode utilizar uma ferramenta *web* para facilitar a criação de gradiente em: [Colorzilla - Gradient editor](#).

11.8 Transform

Através das **CSS3** hoje podemos transformar um elemento, como um *box* e todo o seu conteúdo, utilizando a propriedade **transform**. A propriedade possui algumas variações:

11.8.1 Rotate

É a propriedade **transform** sendo utilizada para girar o elemento. O valor deve ser a quantidade de graus que gostaríamos de girar. Vejamos o exemplo:

```
div{  
    transform: rotate(30deg);  
}
```

A unidade de medida para graus é *deg*, do inglês *degrees* (graus).

11.8.2 Translate

A variação de **transform** para mover o elemento de lugar baseado em sua posição inicial. Vejamos:

```
div {  
    transform: translate(50px,100px);  
}
```

11.8.3 Scale

Variação de **transform** para mudar o tamanho do elemento. O valor será a quantidade de vezes que gostaríamos de aumentar o elemento, por exemplo, o valor dois significa dobrar de seu tamanho original, e assim por diante. Veja o exemplo:

```
div {  
    transform: scale(2,4);  
}
```

11.8.4 Skew

Variação de **transform** para inclinar o elemento. O primeiro valor altera a inclinação no eixo X (horizontal) e o segundo no eixo Y (vertical). Veja:

```
div {  
    transform: skew(30deg,20deg);  
}
```

11.9 Columns

Via CSS3 agora podemos estruturar conteúdos por colunas, normalmente utilizado para textos longos. Ficaria assim:

```
div {  
  columns: 100px 3;  
}
```

No exemplo foi definido o número de colunas e largura das mesmas. Além disso, existem outras propriedades para o uso de colunas, como bordas e espaçamentos, mas a propriedade é pouco usada por haverem poucos cenários para seu uso.

11.10 @Font-face e API Google Webfonts

Um problema que os *web designers* sempre enfrentaram é a obrigação de se utilizar fontes-padrão para a criação dos seus projetos, limitando a estética dos mesmos. Através da propriedade **@font-face** das **CSS3** poderemos utilizar qualquer fonte em nossos projetos, desde que hospedemos a fonte junto com o site no servidor, e ela será automaticamente baixada quando o usuário acessar a página para que ele possa visualizar o conteúdo. Para isso, digite a seguinte configuração em uma seção à parte dentro do código CSS, por exemplo, no início:

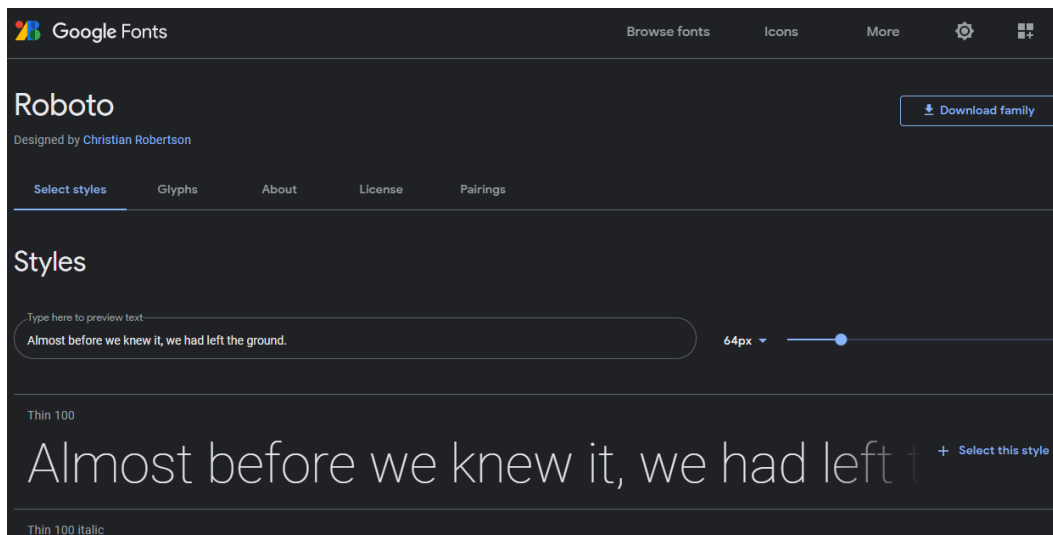
```
@font-face {  
  font-family: "minhafonte";  
  src: url('../fontes/Arista.woff');  
}
```

O efeito é suportado em todos os navegadores e as fontes devem ter a extensão **.woff** ou **.woff2**. Deve-se converter a fonte neste formato em conversores. No momento em que desejar utilizar a fonte específica em um seletor, basta definir o nome dado em **@font-face** como valor da família da fonte:

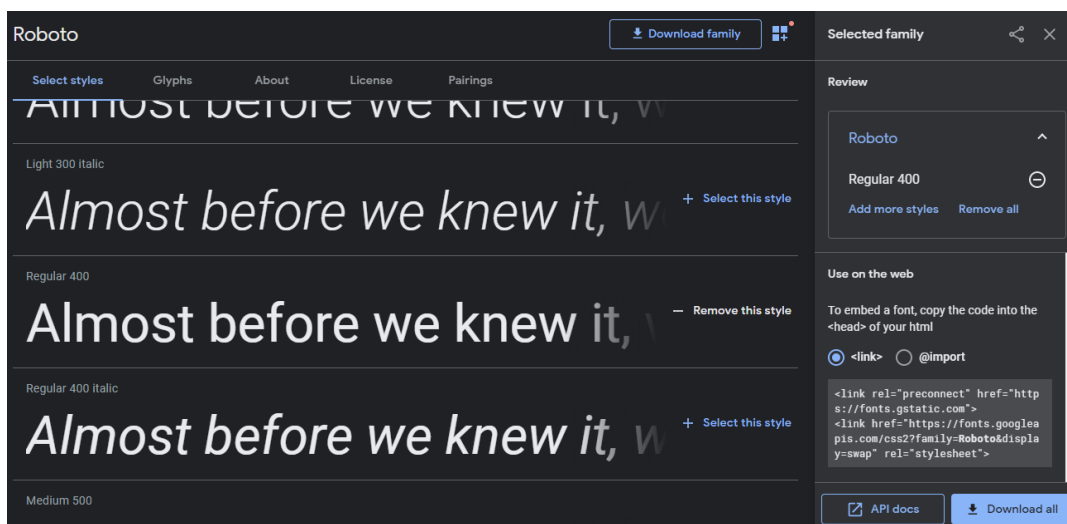
```
h1 {  
  font-family: "minhafonte", Arial, Helvetica, sans-serif;  
}
```

Não esqueça de colocar a fonte dentro da pasta do projeto e indicar o caminho até ela corretamente em **@font-face**, e de hospedá-la no servidor juntamente com os demais arquivos do projeto, para que o texto possa ser aberto corretamente com a fonte desejada.

Outra forma de inserir todo o tipo de fonte em uma página é utilizando a **API** de fontes do **Google**, uma galeria imensa de fontes para *web* onde não há a necessidade de se fazer o *download* da fonte para incorporar ao documento, basta escolher a família desejada e incorporar o *link* ao código. Para isso, acesse a página do [Google Fonts](https://fonts.google.com), pesquise a fonte desejada e clique sobre ela:



Escolha as variações desejadas e clique em “*Select this style*”.



Uma vez definida as fontes desejadas, é gerado um código para ser inserido na seção **head** do documento para que as fontes sejam linkadas a ele.

```
<link rel="preconnect" href="https://fonts.gstatic.com">
<link
href="https://fonts.googleapis.com/css2?family=Roboto&display=swap"
rel="stylesheet">
```

Uma vez linkado ao documento, quando quisermos utilizar a fonte basta definir seu nome na formatação por **CSS** padrão do documento, como fazemos com qualquer outra fonte. Veja:

```
h1 {
  font-family: 'Roboto', sans-serif;
}
```

11.11 Animações via CSS3

Uma das grandes inovações que o **CSS3** trouxe é a possibilidade de criar animações. Elas trazem inúmeras vantagens, são fáceis de fazer e editar, funcionam em qualquer dispositivo e não incorporam muito peso ao desempenho da página. Existem duas formas de fazê-las: através da propriedade **transition** ou através de **@keyframes**:

11.11.1 Transition

Cria um efeito de transição para o elemento. Por exemplo, a animação entre o elemento em seu estado normal, e a transformação que deve ocorrer quando posicionado o mouse sobre ele. Faremos isto através da propriedade **transition**.

```
a {
  font-size: 14px;
  color: #F00;
  transition: 3s ease all;
}
a:hover {
  font-size: 32px;
  color: #900;
  transition: 3s ease all;
}
```

Em seus valores escolhemos onde queremos aplicar a transição. Pode ser definida como **color**, para alterar apenas a cor, ou **font-size** para o tamanho, ou até **opacity** para transparência. No caso definimos **all**, para todos os elementos participarem do efeito de transição. Em um segundo momento, definimos o tempo do efeito em segundos. E por

fim, o tipo de efeito, que possui diversas variações como **ease**, **ease-in**, **ease-out**, **linear**, entre outros.

11.11.2 @keyframes

Keyframes são os chamados quadros-chave, e nos permitem fazer uma animação um pouco mais detalhada. O primeiro passo para criar uma animação por **@keyframes** também partirá do conceito de um estado inicial e um final para animar o elemento, da seguinte forma:

```
@keyframes teste {  
  from {background: red;}  
  to {background: yellow;}  
}
```

Como você pode notar, a regra começa com a declaração de **@keyframes** seguida de um nome qualquer para identificar essa animação, que será necessário no segundo passo. Na sequência, são declarados os atributos **from** (de) e **to** (para), que especificam os estados iniciais e finais da animação respectivamente. Como valores de **from** e **to** foi definido o que eu quero como estado inicial da animação (**from**) e o que eu quero como estado final (**to**), no exemplo, uma mudança de cores.

Também é possível, se preferirmos, definirmos mais estados entre o momento inicial e final. Neste caso não usaremos **from** e **to**, mas faremos via porcentagem, veja:

```
@keyframes teste {  
  0%   {background: red;}  
  25%  {background: yellow;}  
  50%  {background: blue;}  
  100% {background: green;}  
}
```

É possível definir quantas variações de animação quiser, entre os valores 0% e 100%. O segundo passo será atribuir o nome definido na declaração do **@keyframes** ao elemento que gostaríamos de animar, da seguinte forma:

```
div {  
  animation: teste 5s;  
}
```

Veja, especificamos o seletor a ser animado, e atribuímos a ele a propriedade **CSS3 animation**. Como valor deste atributo, definimos o nome da animação **@keyframes** definida anteriormente e a duração do efeito em segundos.

Dica: Se tiver dúvidas sobre quais propriedades **CSS3** você pode utilizar e como anda o suporte dos navegadores, acesse o [Can I Use...?](#)



CAPÍTULO 12

LAYOUT **RESPONSIVO**

Capítulo 12 - O cenário responsivo

Nos anos 90, desenvolver para *web* era muito mais tranquilo no quesito *layout*. Todos (ou a imensa maioria) tinham monitores 14 ou 15 polegadas, em sua resolução 800X600, o que tornava a navegação única para todos os usuários. Bastava definir o tamanho do projeto como 800 *pixels* de largura, e pronto, a página *web* teria uma navegação extremamente semelhante para todos os usuários.

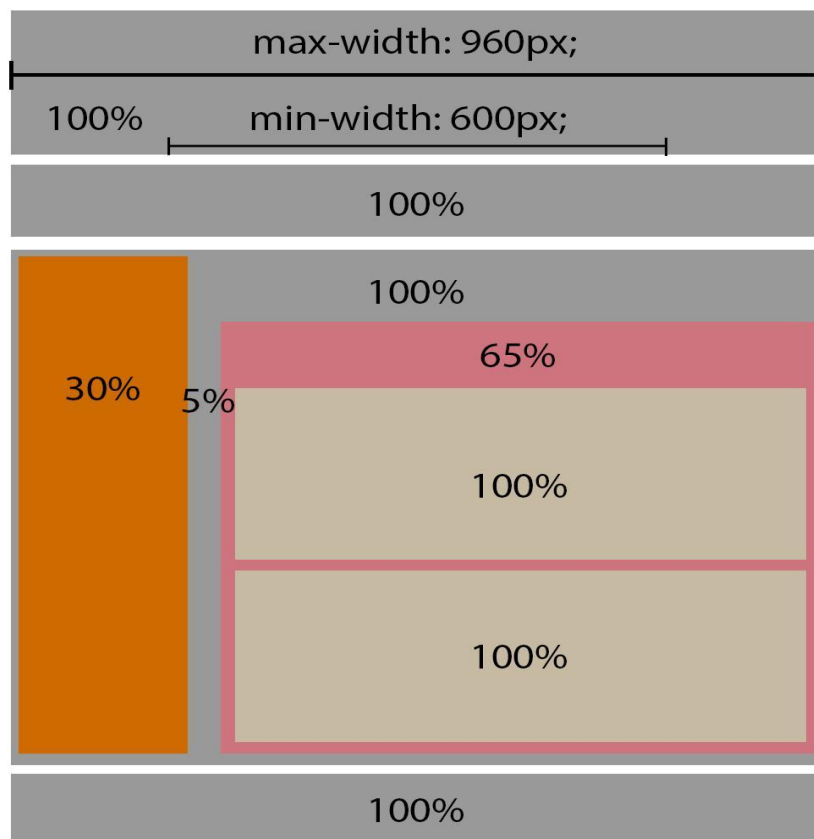
Nos anos 2000 o cenário se complicou um pouco. Com o surgimento de diversos tamanhos de monitores, resoluções *wide-screen* e cada vez maiores, o planejamento do *layout* se tornou um pouco mais cansativo, tendo que se levar em conta muitos fatores. A solução que a maioria optou foi dar uma largura fixa a uma **div** que englobe todo o projeto, geralmente de 960 ou 1024 *pixels*, centralizar a mesma e colocar todo o conteúdo do site dentro daquele espaço. Assim, os poucos usuários que ainda tinham monitores menores teriam uma navegação não ruim, apenas com barra de rolagem horizontal, e a agora grande parte dos usuários, com monitores de 17, 19, 21 polegadas ou maiores teriam uma navegação bem agradável. Essa técnica ainda é válida hoje para todo tipo de site, e pode ser mais aprofundada no site <http://960.gs>, onde é estudada a estruturação de *layout* e o sistema de *grid*, como uma das soluções disponíveis para estruturação do conteúdo da página em colunas.

Em 2007 com a chegada do **iPhone**, seguida do **iPod** com navegador e posteriormente o **iPad**, o cenário tornou-se mais confuso. Quando em 2010 a **Google** também trouxe o seu popular sistema para dispositivos móveis, o **Android**, se tornou imprescindível uma solução para essa gama imensa de tamanhos de telas diferentes por onde os usuários fazem suas navegações pela *web*.

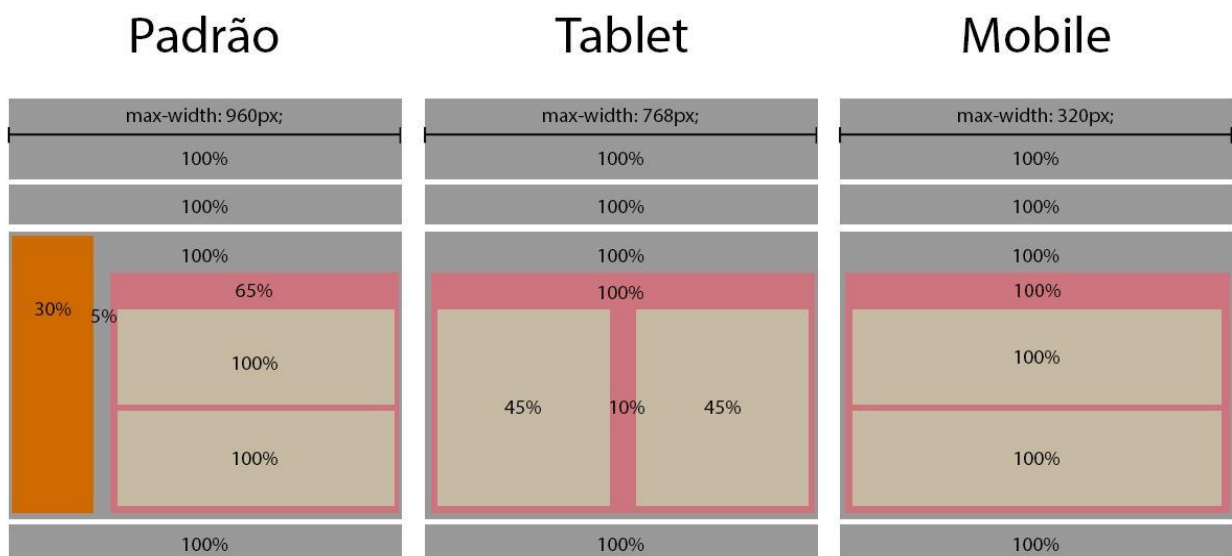
12.1 Layout Responsivo x Layout Fluído

A solução apresentada chegou definitivamente em 2012, com novas **meta-tags**, atributos de *tags* e propriedades **CSS3**, tornou-se possível fazer uma página *web* que se adapte melhor dependendo do dispositivo que a está acessando. Vamos estudar duas técnicas interessantes, estruturação fluída e a responsiva.

Um *layout* estruturado de forma fluída significa que o **CSS** será escrito pensando em uma forma onde o conteúdo da página possa fluir dependendo do tamanho da tela, fazendo com que barras de rolagem horizontais não sejam exibidas, mas que o texto quebre quando a tela for menor, e flua dentro da caixa onde está inserido. No caso de imagens, também é possível fazer com que o seu tamanho se adapte ao tamanho da tela do dispositivo que está acessando o projeto. Essa técnica usa muito dos atributos **CSS** **max-width** e **min-width**, e também atribui vários valores em porcentagem, para que a mesma seja relativa ao tamanho da tela do usuário. Veja o exemplo:



A técnica responsiva é um pouco diferente. Neste caso, teremos uma troca de folha de estilos dependendo do dispositivo que está acessando. Através do que chamamos hoje de **media-queries**, é possível fazer com que quando o usuário acessar o site de um dispositivo móvel, a folha de estilos puxada seja outra, mais adaptada àquele dispositivo. Desta forma, poderemos fazer com que o topo, menu e rodapé não só se adaptem, mas mudem completamente, encaixando perfeitamente em outro dispositivo. Além disso, também colunas de conteúdos, com textos e imagens se adaptarão, para tornar a experiência do usuário na navegação da página uma experiência única e prazerosa.



12.2 Etapas para o layout responsivo

Basicamente, nada foge de tudo o que aprendemos. A inserção de conteúdos, como imagens, *links* e textos permanece a mesma, o conceito de estruturação de *layout* por *divs* ou novas *tags* do **HTML5** também, apenas iremos inserir mais alguns dados e tomar mais cuidado com unidades de medida no **CSS**. Vamos aos passos:

12.2.1 Planejamento

Caso seu cliente queira que o site ou projeto dele seja responsivo, saiba que isso acarretará muito mais horas de trabalho, o que significa que você pode chegar a até dobrar o preço que cobraria pelo projeto não responsivo. Inicialmente, o planejamento será bem maior. Com a infinidade de dispositivos que existem hoje, você poderia ter que planejar como o projeto vai se comportar em cada um deles! Ou mesmo que não fossem tantos, digamos que você irá criar uma versão para computador, outra para **iPhone** e outra para **iPad**, isso significa muito mais tempo de trabalho de um *designer*, que deverá criar todas as telas do projeto para as três versões, além de também muito mais tempo no **HTML/CSS**, para desenvolver o código que irá fazer uma navegação ser perfeita em todos esses dispositivos.

Uma vez tendo consciência de todo o trabalho que dará o desenvolvimento de um projeto responsivo, primeiramente deve-se pensar e planejar como será a navegação em cada um dos dispositivos adaptados (sempre tomando como base a largura de cada um desses dispositivos para planejar). O responsivo deve ter sempre foco no usuário, pensar em como os elementos serão navegados sem mouse, apenas com o *touch* do dispositivo, e como fazer para o usuário ter uma boa experiência com o projeto através de seu dispositivo. Comece planejando no dispositivo móvel e depois vá para o tablet e o computador, para assim não precisar sofrer mais com adaptações posteriores. O **JavaScript** criado não pode ser obstrutivo, ou seja, a página não pode depender do **JavaScript** para funcionar, visto que alguns dispositivos não possuem suporte para o mesmo. É importante no planejamento já ter definido regiões que serão talvez ocultadas caso o site seja acessado de um celular, ou regiões com um conteúdo pesado para ser carregado via 3G.

Em seguida vem a criação do design, que deve ser criado para cada tipo de dispositivo. A utilização do sistema de grid é importante para a criação de um bom *layout* responsivo, e facilita muito na hora de criar o código **CSS**.

12.2.2 Media-queries

Como vimos, existem algumas formas de fazer um site se adaptar a vários dispositivos. Temos o *layout* fluído, trabalhando em porcentagem para que a informação flua de acordo com o *box* que a contém, tornando assim a experiência do usuário um pouco melhor, mas não resolvendo muitos problemas como navegação, por exemplo.

Existe também a forma chamada de *user agent*, onde temos uma linguagem identificando o agente de usuário (o dispositivo e o navegador) através de qual o usuário está acessando a página, e assim redirecionando para um subdomínio, como por exemplo, m.facebook.com.br. Essa técnica não é aconselhada como uma técnica boa para o **SEO**, visto que torna as url's não amigáveis, tendo outro *site* em cada subdomínio, e assim o dispositivo de busca não o verá como um único resultado.

A melhor e mais recomendada forma então veio através de uma implementação da **W3C** para as **CSS3**, chamadas de **media-queries**. Como vimos, o atributo **media** para a tag **link** sempre existiu, mas agora temos novos valores para ela, dependendo do dispositivo. Veja um exemplo:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1">
    <link rel="stylesheet" href="css/style.css" media="screen and
(min-width:960px)">
    <link rel="stylesheet" href="css/tablet.css" media="screen and
(min-width:321px) and (max-width:959px)">
    <link rel="stylesheet" href="css/mobile.css" media="screen and
(max-width:320px)">
    <title>Responsive layout</title>
  </head>
  <body>
    <h1>Conteúdo da página...</h1>
  </body>
</html>
```

IMPORTANTE: Para que as media queries funcionem corretamente, não se esqueça de sempre definir a *meta-tag* **viewport**, fazendo assim com que não ocorra nenhum tipo de alteração no zoom no acesso da página através de dispositivos móveis.

No exemplo, foi linkado uma folha **CSS** para dispositivos com largura mínima de 960 pixels, onde seriam colocados os estilos para computadores e tablets em *landscape* (paisagem), uma outra para tablets em *portrait* (retrato) como intermediária, aqui poderiam haver diversas folhas de estilos com variações para diversos dispositivos e tamanhos de tela diferentes, mais para aprendizado, faremos apenas uma chamada tablet.css, e uma para celulares ou dispositivos entre 0 e 320 pixels de tamanho de tela.

Outra alternativa que trará o mesmo resultado seria utilizar um **@media** dentro de uma folha de estilos, e em seguida dar as condições, como no exemplo:

```
@media screen and (min-width:960px){
  body{
    background: #900;
```

```
}  
}  
@media screen and (min-width:321px) and (max-width:959px) {  
  body{  
    background: #090;  
  }  
}  
@media screen and (max-width:320px) {  
  body{  
    background: #009;  
  }  
}
```

O efeito será o mesmo, neste caso mudando a cor de fundo do projeto dependendo do tamanho da tela. Você poderá redimensionar a janela do seu navegador para ver o efeito de troca de **CSS** acontecer. As vantagens de se usar **@media** é que só é necessário linkar o documento **HTML** em uma folha de estilos, a mesma que dá todas as outras estilizações da página, e dentro dela mesma definir o que muda dependendo de cada dispositivo, fazendo assim com que haja apenas uma requisição no servidor para folhas de estilo. Note que a sintaxe do **CSS** muda um pouco, tendo uma declaração dentro de uma declaração, mas neste caso é comum.

Existem outros valores para o atributo **media**, como **print** para impressão, **orientation**, para a orientação do dispositivo, **resolution**, **aspect-ratio**, entre outros.

Obs.: Esses momentos de troca de folhas de estilo via media queries são conhecidos como *breaking-points*.

IMPORTANTE: Faça seu **CSS** normal para *desktop* e no final do projeto defina as **media-queries**, com seletores iguais, com apenas aquilo que gostaria que fosse diferente em determinados dispositivos.

12.2.3 O conteúdo

Uma vez entendido, planejado, criadas as telas para todos os dispositivos, e feito o uso correto das **media queries** e **viewport**, agora basta escrever o seu **CSS** normalmente, e nos containers, imagens e textos, tomar um pouco de cuidado com as unidades de medida de tudo o que for feito. Por exemplo, se você gostaria que o texto se redimensionasse no tamanho de letra também dependendo do dispositivo, você deveria utilizar uma unidade de medida relativa, como o **em** (tamanho de letra relativa ao elemento pai) ou o **rem** (tamanho de letra relativo ao **body**), para assim, dependendo do tamanho da tela o tamanho da letra também venha a se ajustar. O mesmo vale para imagens, se você gostaria de vê-las ajustadas ao dispositivo utilize tamanhos em porcentagem, e talvez extensões de arquivos vetoriais, como **.svg**, para que a mesma não

perca qualidade no redimensionamento. Existem ótimas calculadoras online para essas conversões como a [PXtoEM](#). Utilize sempre colunas baseadas em um sistema de *grid*, e valores sempre fluídos para as colunas, para facilitar a quebra das mesmas nos *breaking-points*.

Existem conteúdos mais difíceis de adaptar em um projeto responsivo, como formulários, tabelas e *iframes*. No caso de formulários, faça com que o conteúdo seja exibido um embaixo do outro e com campos menores, através das propriedades **CSS display** e **width**. Utilize-se também de formulários semânticos com a *tag label*, para facilitar o acesso aos campos. Para tabelas, é possível ocultar algumas colunas menos importantes ou utilizar a propriedade **overflow:scroll** para ter uma navegação de barra de rolagem dentro apenas daquela região específica. Para *iframes*, utilize também porcentagens, mas tenha em mente que o conteúdo ali exibido não depende de você, caso ele aponte para o site externo, então não será possível adaptá-lo tanto.

12.2.4 Topo, menu e rodapé

É muito importante no *layout* responsivo acontecer a adaptação das áreas constantes do site. No caso do topo, é aconselhável exibir apenas a logo e a cor de fundo, ocultando outros elementos em dispositivos com uma tela pequena. Se no topo houver uma área de busca, mantenha ela fixa no topo da janela para ajudar o usuário a encontrar rapidamente o que precisa na página. Para o menu, faça com que a navegação horizontal se torne vertical, através da propriedade *display*. Se houver menu *popup*, talvez seja necessário algum *plugin jQuery* para um efeito *accordion*. Na área de rodapé é importante quebrar as colunas em apenas uma e ocultar qualquer informação que não seja crucial.

12.3 Aplicações práticas do CSS

Agora que finalizamos a parte de **CSS**, antes de entrar na próxima linguagem, vamos falar sobre algumas coisas que podem te ajudar no dia-a-dia de desenvolvimento de *layouts*.

12.3.1 Reset

O navegador por padrão define diversos estilos aos elementos, chamados de *user agent stylesheets*. Para que isso não venha a te atrapalhar, é importante logo no início do seu arquivo **CSS**, geralmente logo depois do font-face, a definição de um **reset**, que

basicamente é a remoção de estilos padrão do navegador e a padronização de várias partes do projeto. Vejamos um exemplo:

```
/* RESET */
html,
body,
div,
span,
iframe,
h1,
h2,
h3,
h4,
h5,
h6,
p,
a,
address,
em,
img,
strong,
u,
ol,
ul,
li,
fieldset,
form,
label,
legend,
table,
tbody,
tfoot,
thead,
tr,
th,
td,
main,
article,
aside,
figure,
figcaption,
footer,
header,
nav,
section,
time,
audio,
video{
  border:0;
  font-size:100%;
  font:inherit;
  margin:0;
```

```

padding:0;
vertical-align:baseline;
}
html{scroll-behavior:smooth;}
body{
background:#fff;
color:#000;
font-family:"Cera", Arial, Helvetica, sans-serif;
font-size:14px;
line-height:1;
}
ol,
ul,
li{list-style:none;}
strong{font-family:"Cera-bold", Arial, Helvetica, sans-serif;}
table{
border-collapse:collapse;
border-spacing:0;
width:100%;
}
img{width:100%;}
a,
a:hover{
color:#000;
text-decoration:none;
cursor:pointer;
}
:focus{outline:0;}
: hover{transition:.3s ease;}
input[type="text"],
input[type="email"],
input[type="tel"],
input[type="search"],
input[type="url"],
textarea,
select{
border:0;
border-bottom:1px solid #000;
color:#000;
padding:8px 0;
font:14px Arial, Helvetica, sans-serif;
background:transparent;
width:100%;
}
input[type="text"]:focus,
input[type="email"]:focus,
input[type="tel"]:focus,
input[type="search"]:focus,

```

```

input[type="url"]:focus,
textarea:focus,
select:focus{border-color:#900;}
input[type="submit"],
input[type="button"],
button{
  border:1px solid #000;
  text-transform:uppercase;
  padding:10px 30px;
  color:#000;
  font:12px Arial, Helvetica, sans-serif;
  background:transparent;
}
input[type="text"]:placeholder,
input[type="email"]:placeholder,
input[type="tel"]:placeholder,
input[type="search"]:placeholder,
input[type="url"]:placeholder{color:#666;}

```

Geralmente um desenvolvedor sênior tem o hábito de criar seu próprio *reset*, já contendo diversos elementos que possam facilitar o desenvolvimento posterior do projeto. Também nele já é possível entender a fonte do projeto, as cores mais utilizadas, padrões de estilos de formulários e definir isso como ponto inicial, economizando linhas de código.

12.3.2 Menu pop-up

Um comportamento muito comum na maioria dos *sites* é o menu *pop-up*, onde vemos um submenu ao passarmos o mouse sobre um elemento. Vejamos como fazer este comportamento utilizando apenas **HTML** e **CSS**.

```

<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8">
    <title>Menu pop-up</title>
    <style>
      * {
        margin: 0;
        padding: 0;
        border: 0;

```

```

        list-style: none;
    }
    nav {
        background: #900;
    }
    nav>ul {
        display: flex; /* Faz o menu principal ficar com os itens um
ao lado do outro. */
        justify-content: center;
    }
    nav a {
        color: #fff;
        display: block; /* Permite o uso de propriedades de bloco. */
        padding: 10px 30px; /* Aumenta a área de clique, já
centralizando verticalmente e dando altura. */
        text-decoration: none;
    }
    nav li:hover {
        background: #090;
    }
    nav li {
        position: relative; /* Dá a referência para o submenu. */
    }
    nav ul ul {
        position: absolute;
        z-index: 2;
        top: 100%;
        left: 0;
        width: 100%;
        background: #CCC;
        text-align: center;
        display: none;
    }
    nav ul ul a {
        color: #000;
        padding: 10px;
    }
    nav li:hover ul {
        display: block; /* Faz a mágica de passarmos o mouse sobre o
elemento e o submenu aparecer. */
    }
</style>
</head>
<body>
<nav>
<ul>
<li><a href="#">Menu 1</a></li>
<li><a href="#">Menu 2</a></li>
<li>
<a href="#">Menu 3</a>
<ul>
<li><a href="#">Submenu 1</a></li>
<li><a href="#">Submenu 2</a></li>
<li><a href="#">Submenu 3</a></li>

```

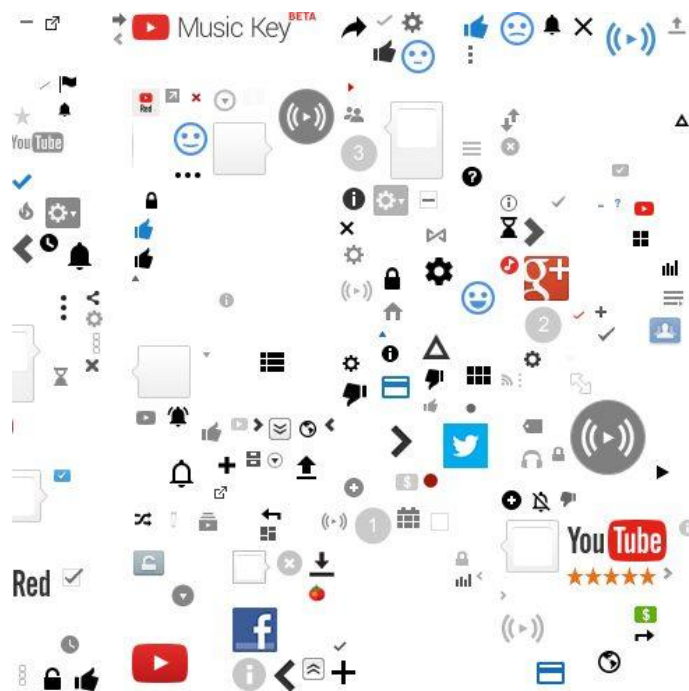
```
        <li><a href="#">Submenu 4</a></li>
      </ul>
    </li>
    <li><a href="#">Menu 4</a></li>
    <li><a href="#">Menu 5</a></li>
  </ul>
</nav>
</body>
</html>
```

12.3.3 DRY

É uma sigla utilizada por alguns desenvolvedores para *Don't Repeat Yourself*, e que faz um trocadilho com a tradução “enxuto”. Resumindo, é o conjunto de diversas técnicas para tentar fazer com que o seu código alcance o mesmo resultado com um menor número de linhas de código. Envolve a técnica você criar classes padrão para elementos que se repetem em alguns casos no *layout*, fazer uma definição boa do projeto já no *reset*, agrupar seletores e propriedades, entre outras. Um código mais enxuto significa um projeto mais rápido e de fácil manutenção.

12.3.4 Sprite

Sprite é o nome de uma técnica CSS que faz o uso de uma única imagem para trazer diversos ícones e imagens do site, geralmente utilizado para aquelas imagens mais constantes do site, como logo, ícones, bandeiras de cartão e afins. Esta técnica torna o *site* mais rápido, pois diminui a requisição de imagens no servidor. Veja o exemplo de uma imagem *sprite*:



A técnica faz uso das propriedades **CSS overflow** e **background-position**. Os valores do **background-position** são negativos, e medidos em um *software* gráfico para exibir apenas aquela área da imagem.

12.3.5 Fontes de símbolos

Outra técnica comum para melhorar a performance do projeto é um uso de fontes de símbolos para a iconografia. Existem algumas famosas e também é possível criar uma própria. Algumas muito comuns de serem utilizadas são a [Font Awesome](#) e a [Glyphicon](#).

12.3.6 Compressão

Alguns projetos utilizam de algumas ferramentas para compressão dos arquivos, removendo comentários e indentação, permitindo assim uma melhor performance e leitura do navegador. Para a manutenção do projeto é importante guardar arquivos descomprimidos.

12.3.7 Pré-processadores

Existem ferramentas que permitem desenvolver o **CSS** de forma mais rápida e prática, são chamadas de pré-processadores de **CSS**. Basicamente possuem outra extensão e necessitam de ferramentas para converter este arquivo em **CSS's** padrões. As mais famosas são **Sass**, **Less** e **Stylus**. O mais famoso é o [Sass Lang](#), ele exige uma configuração mais complicada, através de **Ruby on Rails**, e trabalha com arquivos no formato **.scss**, mas uma vez configurado fica muito mais dinâmico o desenvolvimento das folhas de estilo.



CAPÍTULO 13

INTRODUÇÃO AO **JAVASCRIPT**

Capítulo 13 - Introdução ao JavaScript

Nos anos 90 tivemos início a uma imensa revolução nos meios de comunicação, em específico no que diz respeito à internet. Com a criação de linguagens de marcação como **HTML** e posteriormente linguagens de estilização como **CSS**, pudemos criar páginas *web* completas, com conteúdo e estilização desejada. Através do **HTML** podíamos, e podemos até hoje, inserir imagens, textos, *links*, tabelas, formulários e toda espécie de conteúdo nos nossos projetos. Via **CSS**, podemos alterar toda a estilização e aparência da página, além de estruturá-la através das técnicas de *tableless* aplicadas ao *box-model*. E então, facilmente podemos entregar projetos agradáveis e que satisfazem as necessidades de muitos clientes. O que faltava nesta história era uma interação maior do usuário, com efeitos que chamamos de comportamentos, e através do **JavaScript** podemos fazê-lo.

Linguagem	Função
HTML	Conteúdo da página.
CSS	Aparência e <i>tableless</i> .
JavaScript	Comportamentos.

13.1 A história do JavaScript

JavaScript foi criada em 1995 por *Brendan Eich* da *Netscape* em parceria com a *Sun Microsystems* e implementada em 1996 no navegador *Netscape 2.0*. É uma linguagem desenvolvida para rodar do lado do cliente (*Client-side*) juntamente com **HTML** e **CSS**, ou seja, sua interpretação e funcionamento ocorrem no navegador do usuário, isso ocorre pois existe um interpretador de **JavaScript** hospedado no navegador. Existem ainda as vertentes de **JavaScript** para o lado do servidor (*Server-side*), como o **NodeJS**, mas não trataremos deles aqui. A linguagem foi criada e existe até hoje com a finalidade de gerar interatividade com o usuário, controlar o comportamento do navegador e alterar de forma dinâmica a apresentação dos documentos criados.

Dica: Não confunda **JavaScript** com **Java**, são duas linguagens totalmente diferentes.

Nota: Atualmente, o nome oficial da **JavaScript** é **ECMAScript** e a versão atual desde 2019 é a **ECMA-262 Edição 10**.

Obs.: Muitas vezes iremos nos referir ao **JavaScript** com a abreviatura **JS**, que também é a extensão dos arquivos feitos na linguagem **JavaScript**.

13.2 Boas práticas com JavaScript

Como o **JavaScript** é uma linguagem muito poderosa e abrangente, é necessário tomar alguns cuidados no seu uso, não fazendo apenas por ser possível, mas sim utilizando boas práticas e padrões *web* para que a página possa ter acessibilidade e usabilidade necessárias, além de um melhor desempenho e facilidade de atualização. Confira algumas:

- Não utilizar **JavaScript** na camada errada;

Como vimos, naquilo que chamamos de Front-End (Linguagens que rodam do lado do cliente, como **HTML**, **CSS** e **JS**), temos uma linguagem para cada camada de criação, o **HTML** na camada de conteúdo do projeto web, o **CSS** na camada de apresentação desse projeto e **JS** para comportamentos interativos e dinâmicos. Veja, é possível com **JavaScript** gerar marcação **HTML** e/ou manipular a folha de estilos **CSS** linkada a um documento. Era uma prática comum antigamente utilizarmos **JavaScript** para interferir nas outras camadas do documento, gerando códigos **HTML** e **CSS** indiscriminadamente a partir dos *scripts* de comportamentos, o que fez com o que o **JavaScript** acabasse tendo uma má fama, pois tornava o documento dependente do *script* para ser lido, ou seja, em um dispositivo sem suporte para **JS**, o conteúdo ficava comprometido. Não utilize **JavaScript** para gerar **HTML** e **CSS**, mas deixe que cada camada faça a sua função, não invadindo as outras camadas.

- Não manipule o navegador;

Através de **JavaScript** podemos manipular a janela do navegador, exibindo mensagens de alerta, janelas *pop-up*, alterando suas dimensões, menus e barras. É algo muito invasivo e incômodo na maioria dos casos, então tenha muito cuidado em fazê-lo. Em geral, o intuito do **JavaScript** deve ser enriquecer a experiência do usuário, mas sem atrapalhar a acessibilidade e uso do conteúdo. Resumidamente, cabe ao usuário decidir o que fazer com sua janela do navegador, não aos *scripts* do projeto *web*.

- Não obstrutivo e Melhoria Progressiva.

Segundo os padrões *web*, o **JavaScript** deve se basear em duas práticas de bom uso: ser não obstrutivo e de melhoria progressiva. Não obstrutivo significa que a página funcionará perfeitamente, mesmo que perca alguma usabilidade, caso o usuário acesse a mesma através de um dispositivo sem suporte à **JS**. Resumidamente, a linguagem deve sempre incrementar, não ser decisiva para o conteúdo. O conceito de Melhoria Progressiva (Progressive Enhancement) diz respeito ao primeiro ponto que já debatemos,

o documento deve utilizar todos os recursos do **HTML** primeiramente para se estruturar e definir seu conteúdo, posteriormente deve utilizar tudo o que houver de disponível no **CSS** para incrementar sua página e finalmente o uso de **JavaScript** para melhorar a interatividade e a experiência do usuário. Isso facilitará e muito a manutenção e entendimento dos códigos, além de evitar códigos repetidos e facilitar na correção de eventuais *bugs*.

13.3 Comentários

Para uma melhor compreensão do código inserido, é de bom grado que se comente as regiões do código correspondentes a cada conteúdo. Para comentar o código **JS**, faremos da seguinte forma:

```
// Comentário de linha
/*
  Comentário
  de
  Bloco
*/
```

Podemos comentar qualquer área do código, não prejudicando em nada sua sintaxe ou renderização, pois o navegador sempre irá ignorá-lo.

13.4 A tag <script>

Utilizada para ligar a página **HTML** à uma página de comportamentos **JavaScript**, seja ela local ou em um servidor. Ela também pode ser utilizada para escrever **JavaScript** de forma incorporada ao documento **HTML**, forma não muito aconselhada na maioria dos casos. O código ficaria da seguinte forma:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8">
    <title>Tag Script</title>
  </head>
  <body>

    <script src="js/scripts.js"></script>
  </body>
</html>
```

A ordem faz diferença! No exemplo, é como se os arquivos de JavaScript estivessem sido colocados em uma subpasta chamada “js”. Neste arquivo “scripts.js” você deve escrever todos os comportamentos e efeitos que aprenderemos daqui pra frente. Também não é aconselhável *linkar* um arquivo .js para cada efeito, para não gerar muitas requisições no servidor, assim deixando o seu projeto com pior desempenho para o usuário, mas sim coloque todos os efeitos desejados em um único arquivo de comportamentos e organize esse arquivo com comentários e indentação.

Dica: Por questão de desempenho do seu projeto, você poderá fazer as *tags script* para *linkar* seus arquivos **JS** no final do documento, antes do `</body>`, não prejudicando assim o carregamento do seu projeto com *scripts* muito longos.

13.5 Sintaxe

Diferentemente do **HTML** e **CSS**, a sintaxe do **JavaScript** é muito mais complexa. Basicamente se baseia em definirmos o comportamento desejado no elemento, ou no arquivo **JS**, seguindo o padrão definido pela **W3C** para alcançarmos o comportamento desejado. Basicamente, cada etapa do **JavaScript** terá sua sintaxe, sua forma correta de ser escrita seguindo a lógica desejada. Vamos ver alguns exemplos, só para entendermos esta situação:

```
<body>
  <h2>Meu primeiro JavaScript</h2>
  <button onclick="document.getElementById('demo').innerHTML =
Date()">Clique aqui para a data.</button>
  <p id="demo"></p>
</body>
```

É comum utilizarmos a funcionalidade **alert();** para testarmos se o **JavaScript** está se comportando corretamente, assim:

```
<body>
  <p onclick="funcao1()"></p>
  <script>
    function funcao1(){
      alert("Eu sou um\nAlert!");
    }
  </script>
</body>
```

Também vemos muitas vezes desenvolvedores definindo pontos específicos do documento para visualizar no *console* do navegador. Essa técnica é utilizada normalmente para investigar onde podemos ter um problema no nosso *script*.

```
<body>
  <p onclick="funcao1()"></p>
  <script>
    function funcao1() {
      console.log("Olá Mundo!");
    }
  </script>
</body>
```

Note que sempre finalizamos uma etapa lógica com um ponto e vírgula.

IMPORTANTE: O **JavaScript** é uma linguagem sem muita tolerância. Se houver qualquer erro lógico na construção do *script*, nada além de um erro será renderizado.

Calma, estes foram apenas exemplos para exemplificar a sintaxe ponto a ponto do **JS**, aprenderemos mais sobre estes elementos mais à frente. Por hora, vamos apenas reproduzir estes *scripts* para gerarmos familiaridade.

13.6 Alinhamento de expectativa

Diferentemente das etapas anteriores do curso, onde dominamos **HTML** e **CSS** com exercícios e ensino, no caso de **JavaScript** demanda-se muito mais tempo, estudo, exemplos e aplicações práticas para o seu domínio. Então, neste momento, a real intenção é permitir que você saiba compreender lógicas e saiba ler arquivos **JS** sem dificuldade, além de escrever *scripts* simples. O real domínio dessa linguagem virá com o tempo, com projetos e situações reais do dia a dia, onde você possa partir para o desenvolvimento de *scripts* para conseguir sobrepujar os desafios apresentados.

13.7 Pensamento lógico

O **JavaScript** é uma linguagem de programação. Diferentemente do **HTML** (linguagem de marcação) e do **CSS** (linguagem de estilização), o **JavaScript** exige pensamento lógico para conseguirmos desenvolver o comportamento desejado. Ter um pensamento lógico de baseia em pensar no que se quer, como e quando. Vejamos o seguinte exemplo:

- O que?

Aparecer uma mensagem de alerta.

- Como?

Como uma janela *pop-up* no navegador.

- Quando?

Quando o usuário interage com o botão na tela.

Dito isso, fariamos o *script* seguindo esta lógica básica:

```
<body>
  <p onclick="funcao1()"></p> // Quando
  <script>
    function funcao1(){
      alert("Eu sou um\nAlert!"); // Como, e entre parênteses o O que
    }
  </script>
</body>
```

13.8 Tipos de dados

As informações possuem uma semântica em si do que realmente são. Vejamos os tipos de dados existentes em uma linguagem como o JavaScript:

Tipo de dado	Descrição
Boolean	<i>true</i> e <i>false</i> .
null	Uma palavra-chave que indica valor nulo. Devido JavaScript ser <i>case-sensitive</i> , <i>null</i> não é o mesmo que Null, NULL, ou ainda outra variação.
undefined	Uma propriedade superior cujo valor é indefinido.
Number	42 ou 3.14159.
String	"Olá Mundo!"
Symbol	(novo em ECMAScript 6). Um tipo de dado cuja as instâncias são únicas e imutáveis.

Além disso, também temos **Objetos** e **funções**. Você pode pensar em objetos como recipientes para os valores, e funções como métodos que suas aplicações podem executar.

13.9 Variáveis

Como muitas vezes no **JavaScript** pode se tornar repetitivo o uso do mesmo código inúmeras vezes, podemos definir variáveis, como objetos a serem consumidos em outros momentos do código, da seguinte forma:

```
var myObj = { nome: "John", sobrenome: "Doe" };  
console.log(myObj);
```

Observe que a variável inicia-se com o termo **var**, seguida de uma **string** (termo genérico sem caracteres especiais) para nomear a variável, seguida da declaração do seu valor. Por fim, chamamos a variável sempre que desejarmos consumi-la.

Dica: Quando definimos diversos valores para um mesmo elemento, separados por vírgula, chamamos isso de **array**.

13.10 Operadores

No desenvolvimento de lógicas **JavaScript** precisaremos inúmeras vezes de operadores, como sinais matemáticos para a definição do que se deseja. Veja alguns operadores mais comuns:

Operador	Explicação	Simbolo(s)	Exemplo
Adição	Usado para somar dois números ou juntar duas strings (concatenação).	+	6 + 9; "Ola " + "mundo!";
Subtração, multiplicação, divisão	Fazem exatamente o que você espera que eles façam na matemática básica.	-, *, /	9 - 3; 8 * 2; 9 / 3;
Atribuição	Define que algo vale determinado valor.	=	let minhaVariavel = 'Bob';
Igualdade	Faz um teste para ver se dois valores são iguais, retornando um resultado <i>true/false</i> (booleano).	===	minhaVariavel === 4;
Negação, não igual	Quando usado junto com o operador de igualdade, o operador de negação testa se os valores são diferentes.	!, !==	minhaVariavel !== 3;
Incremento	Atribui valores até determinado ponto definido.	++	x = ++y
Decremento	Remove valores até determinado ponto definido.	--	x = --y
Ou	Define que um ou outro valor podem ser verdadeiros ou falsos.		(x === 5 y === 5) is false
E	Define que um e outro valor podem ser verdadeiros ou falsos.	&&	(x < 10 && y > 1) is true

Os operadores sozinhos podem não trazer resultado, mas dentro de uma lógica **JavaScript** podem ser muito úteis.

13.11 Condições

Em uma lógica de programação, muitas vezes vemos condicionais, que são definidos pela palavra chave **if** e **else** (se, então). Basicamente definem um argumento lógico condicional que se atendido, retorna o valor desejado. A sintaxe seguirá a seguinte lógica:

```
if (condição) {  
  instrução1  
} else {  
  instrução2  
}
```

Vejam os exemplos de uma condicional:

```
<body>  
  <p>Clique para ter uma saudação:</p>  
  
  <button onclick="myFunction()">Clique aqui</button>  
  <p id="demo"></p>  
  <script>  
    function myFunction() {  
      var hour = new Date().getHours();  
      var greeting;  
      if (hour < 18) {  
        greeting = "Bom dia!";  
      } else {  
        greeting = "Boa noite!";  
      }  
      document.getElementById("demo").innerHTML = greeting;  
    }  
  </script>  
</body>
```

Em lógicas mais complexas de condicionais também existem os termos **else if** e **switch**. Veja um exemplo:

```
<body>  
  <p id="demo"></p>  
  <script>  
    var day;  
    switch (new Date().getDay()) {  
      case 0:  
        day = "Domingo";  
        break;  
      case 1:  
        day = "Segunda";  
    }  
  </script>  
</body>
```

```
        break;
    case 2:
        day = "Terça";
        break;
    case 3:
        day = "Quarta";
        break;
    case 4:
        day = "Quinta";
        break;
    case 5:
        day = "Sexta";
        break;
    case 6:
        day = "Sábado";
    }
    document.getElementById("demo").innerHTML = "Today is " + day;
</script>
</body>
```

O **switch** é menos utilizado, mais em casos de uma lógica que exija uma análise de diversos elementos.

13.12 Laços

Laços (*loops*) servem para executar um bloco de código determinado número de vezes. Para isso utilizamos **for** ou **while**. Vejamos:

```
<body>
  <p id="demo"></p>
  <script>
    var text = "";
    var i;
    for (i = 0; i < 5; i++) {
      text += "O número é " + i + "<br>";
    }
    document.getElementById("demo").innerHTML = text;
  </script>
</body>
```

Como resultado teremos uma listagem de números até 5. Veja que na declaração do **for** temos o valor inicial, o ponto de parada e por fim o incremento. Podemos fazer uma lógica semelhante com **while**:

```
<body>
  <p id="demo"></p>
  <script>
    var text = "";
    var i = 0;
    while (i < 5) {
      text += "<br>O número é " + i;
      i++;
    }
    document.getElementById("demo").innerHTML = text;
  </script>
</body>
```

No **while** a sintaxe muda um pouco, mas a ideia é a mesma. Fique atento para sempre definir um fim para seu laço, ou pode gerar um *bug* lógico.

13.13 Funções

Função no **JavaScript** é o nome dado para um bloco de código para uma tarefa específica. Geralmente definimos um objeto lógico dentro de uma função para que este mesmo seja chamado em um momento específico. Funções podem ser anônimas, quando não declarado seu nome, apenas a lógica, ou nominais, com o nome declarado para futura chamada. Veja um exemplo:

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

13.14 Debugando

Em diversos momentos o **JavaScript** nos retornará erros e precisaremos entender o problema apresentado e buscar entender o que aconteceu para resolvermos o problema. Vejamos algumas dicas:

- Utilize o *console* do navegador para fazer testes;
- Utilize **console.log** para tentar entender em qual momento o script quebrou;
- Ponto e vírgula e outras coisas simples muitas vezes são a causa do erro;
- Entenda a causa do erro para encontrar a solução. Ele pode ser de:
 - Sintaxe - Faltou algo na hora de escrever o código;
 - Lógica - A lógica em si não faz sentido;
 - Referência - Falta um objeto ou elemento referenciado.

Veja como interpretar os erros mais comuns do **JavaScript**:

Tipo de dado	Descrição
NaN	<i>Not A Number.</i>
undefined	Elemento não teve uma igualdade atribuída.

null	Representa um valor nulo ou vazio.
UNCAUGHT TYPEERROR: CANNOT READ PROPERTY	Este erro ocorre quando você quer acessar uma propriedade ou método de um objeto que não existe.
TYPEERROR: 'UNDEFINED' IS NOT AN OBJECT	Este erro é o mesmo que foi descrito acima, a diferença é que esta mensagem de erro aparece no Chrome.
TYPEERROR: 'THIS.<>' IS NOT A FUNCTION	Este erro ocorre quando você chama uma função que não foi definida.
RANGEERROR	Ocorre quando um número informado não é compatível com o intervalo de números válidos.
REFERENCEERROR	Os erros de referência ocorrem quando ocorre uma referência inválida.
SYNTAXERROR	Ocorrem quando é detectado um erro de sintaxe na programação.
TYPEERROR	Estes erros ocorrem quando um valor tem um tipo diferente daquele que é esperado.



CAPÍTULO 14

INTRODUÇÃO AO JQUERY

Capítulo 14 - Introdução à biblioteca jQuery

A biblioteca **jQuery** foi criada por John Resig em 2006, em um blog do autor. Nas palavras do autor: “O foco principal da biblioteca **jQuery** é a simplicidade. Por que submeter os desenvolvedores ao martírio de escrever longos e complexos códigos para criar simples efeitos?”. Resumidamente, é uma maneira mais fácil de escrever **JavaScript**, não precisando ser um programador experiente para conseguir colocar em uso efeitos bonitos e dinâmicos em páginas *web*. Através desta biblioteca podemos usar efeitos visuais e animações, prover interatividade, trazer informações sem o recarregamento da página, alterar, modificar e simplificar a apresentação e estilização da página.

A biblioteca **jQuery** é disponibilizada como *software* livre e aberto, e seu uso é regido conforme as regras do MIT (*Massachusetts Institute of Technology*) e pelo GPL (*GNU General Public License*), ou seja, você pode usar a biblioteca para projetos pessoais e comerciais.

14.1 Diferenças entre JavaScript e jQuery

jQuery é **JavaScript**. As principais diferenças estão na sintaxe (modo de escrever a linguagem) e no método de trabalho. Enquanto via **JavaScript** seriam necessárias muitas linhas de código para conseguir um efeito comum de se utilizar na *web*, utilizando a biblioteca **jQuery** podemos fazer o mesmo com muito menos linhas de código. A vantagem do **JavaScript** “puro” é que podemos fazer muito mais e sem *linkarmos* o documento a nenhuma biblioteca. Já no **jQuery**, temos algumas limitações, e temos que gerar o vínculo na biblioteca, mas a facilidade que isto traz, na economia de códigos e simplicidade na geração dos comportamentos mais comuns costuma fazer valer o custo.

14.2 Conhecendo jQuery

Como vimos, é interessante fazer uso de **JavaScript** em seus projetos devido a necessidade de colocar comportamentos e interações com o usuário que enriqueçam a experiência do mesmo. Mas devido ao **JS** ser uma linguagem muito abrangente, havia a necessidade de um programador experiente para criar essas interações. Através da biblioteca **jQuery** isso se tornou muito mais fácil e rápido, e com o uso de seus *plugins*, não é necessário programar para conseguir fazer os efeitos mais interessantes funcionarem.

14.3 Baixando e linkando bibliotecas

jQuery não é a única biblioteca **JavaScript**, mas com certeza é hoje a mais famosa. Na verdade, o **jQuery** na prática nada mais é do que um arquivo **JavaScript**, com a extensão .js que podemos, ou baixar e *linkar* no nosso documento localmente, ou utilizar versões *online* disponibilizadas pela **Google** e **Microsoft** por exemplo. Uma vez feita a ligação com a biblioteca, é interessante ter um outro arquivo para escrever seus *scripts JS*, como um chamado “scripts.js” por exemplo, para este arquivo escrever as chamadas dos efeitos que gostaria de efetuar no seu projeto. É importante salientar que neste arquivo scripts.js você pode escrever tanto na linguagem **JavaScript**, quanto na sintaxe da **jQuery**, desde que tenha feito o *link* na biblioteca.

14.4 Sintaxe das linguagens

JavaScript tem um jeito todo peculiar de ser escrito, que é um pouco diferente de quando o utilizamos através de **jQuery**. Claro que o intuito aqui não é que você escreva um arquivo **JS** manualmente, mas é interessante conhecer um pouco do que está escrito nos *scripts* para poder manipulá-lo posteriormente. Veja o seguinte exemplo:

Sintaxe JavaScript	Sintaxe jQuery
document.getElementsByTagName("p")	\$("p")
document.getElementById("um").setAttribute("class", "cor")	\$("#um").attr("class", "cor")

Perceba a simplicidade do uso da biblioteca **jQuery**. Note, nos dois momentos foi escrito a mesma coisa, mas com **jQuery** podemos escrever muito menos para conseguir o mesmo resultado. Então, conforme já foi dito, **jQuery** é **JavaScript**, mas uma vez *linkada* na biblioteca, esse **JavaScript** pode ser escrito de forma muito mais simples e objetiva.

Outro detalhe a se prestar atenção no exemplo anterior e que ajuda muito quem já tem conhecimento de **CSS**: o **jQuery** utiliza os mesmos seletores das folhas de estilo, podendo agregar um conhecimento ao outro.

Veja também que interessante o fato do uso do cifrão (\$) na sintaxe **jQuery**. O cifrão seguido de parênteses no **jQuery** é o que chamamos de “função construtora”, e estará presente em todos os scripts que vier a escrever, é como uma solicitação, capturando o seletor que receberá um comportamento. Outras bibliotecas também usam o cifrão, então para evitar conflitos caso utilize outras bibliotecas, substitua-o por **jQuery()**. Resumindo:

Este código **HTML**:

```
<h1 id="titulo1">Texto teste</h1>
```

Pode ser selecionado via JavaScript assim:

```
document.getElementById("titulo1")
```

E via jQuery assim:

```
$('#titulo1')
```

IMPORTANTE: Você ainda pode utilizar os seletores avançados e de **CSS3** no **jQuery** que eles funcionarão.

14.5 A Biblioteca na prática

Aqui veremos como baixar, *linkar* e utilizar alguns efeitos da biblioteca **jQuery**, assim como instalar os principais *plug-ins*. **jQuery** pode ser escrito em qualquer editor de texto, assim como o **HTML** e o **CSS** do seu projeto.

14.5.1 Linkando

Como vimos, podemos utilizar a biblioteca **jQuery** de forma local ou *linkada* em servidores onde ela já esteja hospedada, aqui no caso utilizaremos a biblioteca hospedada no próprio servidor do **jQuery**, no entanto, vamos aprender a fazer localmente caso queira. Para isso, acesse o site oficial da [jQuery](#) e clique no botão para *download*. Em seguida poderemos escolher a versão para utilizar. Clique no link “*Download the compressed, production jQuery*”. Ao clicar no link, alguns navegadores farão o *download* do arquivo automaticamente, outros abrirão uma tela com o código fonte da biblioteca, no primeiro caso apenas coloque o arquivo na pasta do seu projeto, já no segundo, copie o código, coloque em seu editor de códigos e salve com um nome apropriado, como por exemplo “jquery.min.js”, que é o nome que aparece na url, e coloque o arquivo na pasta do seu projeto.

Obs.: “*compressed*” significa que a biblioteca é comprimida, ou seja, não há comentários e nem indentação, é a melhor opção para quem quer apenas *linkar* na

biblioteca. A versão “*uncompressed*” contém comentários e indentação para o estudo da biblioteca em si.

Nota: Algumas versões podem não ser compatíveis com alguns *plug-ins*, sempre verifique na descrição do *plug-in*.

Pronto! Feito isso você já tem a biblioteca **jQuery** em um arquivo na pasta do seu projeto. Note que ele é realmente apenas um arquivo escrito em **JavaScript**. Agora basta apenas fazer a ligação entre o seu projeto e a biblioteca. Como comentado, no nosso caso utilizaremos a biblioteca hospedada no site do **jQuery**, que pode ser encontrada no site do [jQuery](#). Veja como ficaria o código:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8">
    <title>jQuery</title>
  </head>
  <body>

    <script src="https://code.jquery.com/jquery-3.6.0.min.js"
integrity="sha256-/xUj+3OJ+30JU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4="
crossorigin="anonymous"></script>
    <script src="js/scripts.js"></script>
  </body>
</html>
```

Dica: Apenas *link* na biblioteca e não a altere jamais. Tome cuidado também para *linkar* na biblioteca apenas uma vez e ter apenas uma versão dela no seu projeto, para evitar conflitos.

14.5.2 Construindo uma função

Vimos que a sintaxe **jQuery** é muito semelhante à do **JavaScript**, só que mais simples, vamos analisá-la. Quando escrevemos nosso documento **jQuery**, é importante começar sempre os *scripts* com “**\$(document).ready(function(){});**”, para que a função seja trazida apenas após o carregamento completo do documento. Depois, entre as chaves, inserimos um seletor, que irá indicar o elemento **HTML** a ser atingido. E finalmente, após um ponto “.”, inserimos a função, método ou efeito **jQuery** que gostaríamos de realizar, como no exemplo:

```
<body>
  <p>Clique Esconder para ocultar o texto</p>

  <script src="https://code.jquery.com/jquery-3.6.0.min.js"
integrity="sha256-/xUj+3OJU5yExlq6GSYGGShk7tPXikynS7ogEvDej/m4="
crossorigin="anonymous"></script>
  <script>
    $(document).ready(function() {
      $("p").click(function() {
        $(this).hide();
      });
    });
  </script>
</body>
```

Note que utilizamos a palavra chave **this**, para que o próprio elemento selecionado receba o efeito do comportamento.

Nota: Verifique no final da apostila a tabela com os métodos e efeitos **jQuery**.

Em momentos onde haja diversas declarações de comportamentos no projeto, utilize um único “**\$(document).ready(function(){});**” para conter todos os comportamentos dentro dele. Assim como no **JavaScript**, funções podem ser anônimas ou nominais, dependendo da necessidade.

Importante: Funções nominais não devem ficar dentro do **\$(document).ready**.

Veja um exemplo de como a sintaxe **jQuery** funciona:



A função construtora “chama” um elemento a ser selecionado do **HTML** para receber comportamentos. Quando se trata de uma palavra-chave conhecida pelo **jQuery** não utilizamos aspas, quando se trata de outros tipos de seletores, utilizamos aspas. Os métodos e efeitos **jQuery** definem o que deve acontecer. Por fim, enclausuramos em funções para que o **jQuery** entenda os blocos a serem estilizados.

14.6 Efeitos da biblioteca jQuery

Agora estamos configurados para começarmos a trabalhar. É bom também *linkarmos* sempre em um arquivo **CSS**, pois alguns efeitos também exigirão estilização para funcionar. O **jQuery** possui vários efeitos que podem ser utilizados sem a necessidade de *plug-ins*, mas como nosso objetivo não é se aprofundar em programação, vejamos os mais simples que podem ser necessários no cotidiano de um projeto *web*:

Nota: Para fins de estudos, os códigos exibidos aqui terão seus estilos e comportamentos incorporados no próprio documento **HTML**, para que fique mais fácil de ser visualizado. Entretanto, em seus exercícios e projetos sempre utilize um arquivo *linkado* para estilos e outro para comportamentos.

14.6.1 Esconder e exibir elementos

Um efeito muito comum nos sites é o surgimento e a ocultação de elementos após a interação do usuário através do clique, isso pode ser facilmente realizado via **jQuery**. Utilizaremos, por exemplo, o seletor de ID “\$(“#hide”)” e que ao ser clicado “.click(function()” deve executar a ocultação de todo elemento **<p>** do documento **HTML** “\$(“p”).hide();”. Posteriormente, será feito a mesma coisa, mas com o seletor sendo o ID “\$(“#show”)”, que quando clicado, fará com que o elemento **<p>** apareça “\$(“p”).show();”. Resumindo, com **jQuery** temos algumas classes prontas para ocultar e exibir elementos no seu projeto, o .hide(); e o .show();. Tal simples efeito de ocultação e exibição é comum de ser feito facilmente via **CSS**, mas para que aconteça como no exemplo, ou seja, no momento do clique em outro elemento, é necessário algum tipo de comportamento.

```
<body>
  <p>Clique Esconder para ocultar o texto</p>
  <button id="hide">Esconder</button>
  <button id="show">Mostrar</button>

  <script src="https://code.jquery.com/jquery-3.6.0.min.js"
integrity="sha256-/xUj+3OJU5yExlq6GSYGGShk7tPXikynS7ogEvDej/m4="
crossorigin="anonymous"></script>
  <script>
    $(document).ready(function () {
      $("#hide").click(function () {
        $("p").hide();
      });
      $("#show").click(function () {
        $("p").show();
      });
    });
  </script>
</body>
```

Podemos fazer os efeitos de ocultar e exibir com um único efeito, chamado **toggle**. Para fazer essa ocultação/exibição com efeito, utilize “.toggle(“slow”);”, como no exemplo:

```
<body>
  <div id="flip">Clique aqui!</div>
  <div id="panel">Olá Mundo!</div>

  <script src="https://code.jquery.com/jquery-3.6.0.min.js"
integrity="sha256-/xUj+3OJU5yExlq6GSYGGShk7tPXikynS7ogEvDej/m4="
crossorigin="anonymous"></script>
  <script>
    $(document).ready(function () {
      $("#flip").click(function () {
```

```

        $("#panel").toggle("slow");
    });
});
</script>
</body>

```

O valor da função pode ser “slow”, “fast” ou milissegundos desejados para a animação.

14.6.2 Manipular HTML

Apesar de não ser uma boa prática misturar as camadas de desenvolvimento, às vezes a única saída para o efeito desejado é inserir **HTML** ou manipular atributos de *tag* e valores de atributos através do **jQuery**. Vejamos um exemplo:

```

<body>
  <p>Texto exemplo.</p>
  <button>Clique aqui</button>

  <script src="https://code.jquery.com/jquery-3.6.0.min.js"
integrity="sha256-/xUj+3OJU5yExlq6GSYGGSHk7tPXikynS7ogEvDej/m4="
crossorigin="anonymous"></script>
  <script>
    $(document).ready(function () {
      $("button").click(function () {
        $("p").html("<span class='red'>Hello <b>Again</b></span>");
      });
    });
  </script>
</body>

```

Note o efeito `.html()`; manipulando o **HTML** do documento. Também podemos trabalhar nos atributos de *tag* com o efeito `.attr()`;

```

<body>
  
  <button>Clique aqui</button>

  <script src="https://code.jquery.com/jquery-3.6.0.min.js"
integrity="sha256-/xUj+3OJU5yExlq6GSYGGSHk7tPXikynS7ogEvDej/m4="
crossorigin="anonymous"></script>
  <script>
    $(document).ready(function () {
      $("button").click(function () {
        $("img").attr("alt", "Descrição da imagem");
      });
    });
  </script>

```

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

14.6.3 Atribuir estilos

Como já foi dito, inserir **HTML** e **CSS** via **JS** não é uma boa prática, mas no caso do **CSS** as vezes é necessário alterar alguma estilização para um efeito funcionar, não prejudicando a estrutura do projeto.

```
<body>
  <h2>Texto exemplo</h2>
  <p>Texto exemplo.</p>
  <button>Definir estilos para p</button>

  <script src="https://code.jquery.com/jquery-3.6.0.min.js"
integrity="sha256-/xUj+3OJU5yExlq6GSYGGShk7tPXikynS7ogEvDej/m4="
crossorigin="anonymous"></script>
  <script>
    $(document).ready(function () {
      $("button").click(function () {
        $("p").css({ "background-color": "yellow", "fontsize": "18px"
});
      });
    });
  </script>
</body>
```

Veja a propriedade “.css” em ação, utilizando as mesmas propriedades **CSS** em seus valores, entre aspas e separados por vírgula.

Nota: Para mais ações via **jQuery**, confira [W3Schools - jQuery](#) ou na sua documentação [API jQuery](#).

Outra forma de atribuírmos estilos é através da adição de classes via **jQuery**, da seguinte forma:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8">
    <title>jQuery</title>
    <style>
      .teste {
        background-color: yellow;
        font-size: 18px;
```

```
    }  
    </style>  
</head>  
<body>  
    <h2>Texto exemplo</h2>  
    <p>Texto exemplo.</p>  
    <button>Definir estilos para p</button>  
  
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"  
integrity="sha256-/xUj+3OJU5yExlq6GSYGGShk7tPXikynS7ogEvDej/m4="</script>  
crossorigin="anonymous"></script>  
    <script>  
        $(document).ready(function () {  
            $("button").click(function () {  
                $("p").addClass("teste");  
            });  
        });  
    </script>  
</body>  
</html>
```

Note que para adicionar classes utilizamos o efeito `addClass()`. O nome da classe deve vir entre aspas, sem a necessidade da declaração do ponto “.”. A classe deve estar definida no **CSS** para aplicar o efeito desejado.

Também existe o efeito `removeClass()`; para remover classes na interação, e `toggleClass()`, para atribuir e remover classes em um único efeito.



CAPÍTULO 15

PLUGINS JQUERY

Capítulo 15 - Plugins jQuery

Podemos chamar de *plug-in* **jQuery** algum efeito que tenha sido feito por algum desenvolvedor e disponibilizado para *download* e uso da comunidade. Há alguns pagos, outros gratuitos, vejamos alguns mais comuns de serem vistos e usados nos principais projetos *web*.

15.1 Lightbox

Lightbox foi o primeiro *plug-in* para **jQuery**! É um *plug-in* utilizado para exibição de imagens em uma melhor resolução em uma janela modal quando clicado nas miniaturas (*thumbnails*). Fácil de instalar e bem documentado, o Lightbox é amplamente utilizado em sites, principalmente para melhor visão de imagens. Para instalar ele em seu projeto, primeiramente entre no site [Lokeshdhakar - lightbox](#).

Nota: A maioria dos *plug-ins* atuais também oferecem sua versão no **GitHub**. É uma forma de hospedar o *plug-in* de forma compartilhada, para que a comunidade possa ajudar na construção de melhorias para o código e sugeri-las ao autor.

Clique em *download* para baixar os arquivos do *plug-in*. Eles virão compactados, mas podem ser descompactados e incorporados nos arquivos do seu projeto.

Dica: Como dito anteriormente, é interessante que você coloque os códigos dos arquivos baixados dos *plug-ins* dentro dos arquivos de estilos e comportamentos do seu projeto, de forma comentada e indentada para melhor organização.

Em seguida, construa uma galeria de imagens em **HTML**, inserindo um *link* em cada imagem para ela mesma, ou para sua versão maior, como no exemplo:

```
<body>
  <ul>
    <li><a href="imagens/imagem1.jpg"></a></li>
    <li><a href="imagens/imagem2.jpg"></a></li>
    <li><a href="imagens/imagem3.jpg"></a></li>
    <li><a href="imagens/imagem4.jpg"></a></li>
```

```

    <li><a href="imagens/imagem5.jpg"></a></li>
  </ul>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"
integrity="sha256-/xUj+3OJU5yExlq6GSYGSXk7tPXikynS7ogEvDej/m4="
crossorigin="anonymous"></script>
  <script src="js/scripts.js"></script>
</body>

```

Abaixo no site você ainda pode conferir como funciona o efeito, e os passos para instalá-lo. É simples, uma vez colocado o conteúdo do arquivo **CSS** e **JS** do *plug-in* nos arquivos do seu site, basta alterar o **HTML**, colocando o ativador em cada *link*, assim:

```

<body>
  <ul>
    <li><a href="imagens/imagem1.jpg" datalightbox="roadtrip"></a></li>
    <li><a href="imagens/imagem2.jpg" datalightbox="roadtrip"></a></li>
    <li><a href="imagens/imagem3.jpg" datalightbox="roadtrip"></a></li>
    <li><a href="imagens/imagem4.jpg" datalightbox="roadtrip"></a></li>
    <li><a href="imagens/imagem5.jpg" datalightbox="roadtrip"></a></li>
  </ul>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"
integrity="sha256-/xUj+3OJU5yExlq6GSYGSXk7tPXikynS7ogEvDej/m4="
crossorigin="anonymous"></script>
  <script src="js/scripts.js"></script>
</body>

```

O termo “roadtrip” é opcional, e serve apenas para mostrar a qual galeria de imagens essa imagem especificamente pertence. Se quiser traduzir o efeito, você encontra os termos no arquivo **JS**, e para alterar a aparência, você encontra o que for necessário no arquivo **CSS**.

15.2 Slider e Carousel

Slider é um efeito conhecido por ser utilizado na área de banners em destaque dos principais sites. Existem vários *plug-ins* ou formas de fazer esse tipo de efeito, assim como para um programador experiente em **JS** não exige muito para desenvolvê-lo manualmente. Geralmente se baseia em uma lista de marcadores com imagens no documento **HTML**, alguma estilização baseando-se na propriedade **overflow** em **CSS**, e o comportamento de deslocamento ao clicar nas setas via **JS**. Aqui iremos utilizar um *plug-in* chamado **Slick** para desenvolver o efeito, ele é simples e funciona bem em todos os navegadores.

Aqui você poderá ver um exemplo do *plug-in* funcionando tanto para Slider quanto para Carousel, que veremos a seguir. Primeiramente, devemos fazer o *download* dos arquivos necessários para o *plug-in* funcionar. Você poderá baixar os arquivos e conferir toda a documentação no [Github Ken Wheeler](#). Basicamente, uma vez estando com o **JS** e **CSS** com o conteúdo do *plug-in*, basta preparar o **HTML**, com o ativador do efeito em **JavaScript** no final, ou no arquivo “.js”, assim:

```
<body>
  <article>
    <ul class="slider">
      <li></li>
      <li></li>
      <li></li>
    </ul>
  </article>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"
integrity="sha256-/xUj+3OJU5yExlq6GSYGSXh7tPXikynS7ogEvDej/m4="
crossorigin="anonymous"></script>
  <script>
    $(document).ready(function () {
      $('.slider').slick({
        dots: true,
        infinite: true,
        slidesToShow: 1,
        slidesToScroll: 1,
        speed: 500,
        fade: true,
        cssEase: 'linear',
        prevArrow: '<i class="fas fa-angle-left left"></i>',
        nextArrow: '<i class="fas fa-angle-right right"></i>'
      });
    });
  </script>
</body>
```

Perceba na documentação que é possível customizar o **Slick** de diversas formas possíveis, mas ele sempre irá se basear no ativador do container pai definido no arquivo **JS**. Utilize os campos definidos para customização das setas, e o **CSS** para estilização dos *dots* de passagem de slide. Para realizar o efeito em carrossel e prevendo comportamentos de responsividade, confira o código de exemplo:

```
<body>
  <article>
    <ul class="carousel">
      <li></li>
      <li></li>
      <li></li>
      <li></li>
      <li></li>
    </ul>
  </article>
```

```

    <li></li>
  </ul>
</article>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"
integrity="sha256-/xUj+3OJU5yExlq6GSYGGShk7tPXikynS7ogEvDej/m4="
crossorigin="anonymous"></script>
<script>
  $(document).ready(function () {
    $(".carousel").slick({
      dots: false,
      infinite: true,
      speed: 1000,
      autoplay: true,
      autoplaySpeed: 4000,
      arrows: true,
      slidesToShow: 4,
      responsive: [
        {
          breakpoint: 991,
          settings: {
            slidesToShow: 1
          }
        }
      ],
      prevArrow: '<i class="fas fa-angle-left left"></i>',
      nextArrow: '<i class="fas fa-angle-right right"></i>'
    });
  });
</script>
</body>

```

15.3 Formulários

Formulários necessitam diversos *scripts* para comportamentos de validações e de garantir que o comportamento desejado seja atendido. Vejamos alguns.

15.3.1 Validação

Validar um formulário é fazer com que alguns campos tenham algum tipo de conferência se os dados digitados ali estão corretos com o que é esperado ou não, por exemplo, pedir que campos sejam obrigatórios, outros sejam endereços de e-mail, números, e assim por diante. Como foi visto, em **HTML5** existem alguns novos atributos de *tags* que já fazem alguma validação, mas para que funcione corretamente em todos os navegadores e para que tenhamos mais possibilidades de personalização, iremos ver como fazer isso via **jQuery**.

Temos vários *plug-ins* diferentes para fazer validação e também há a possibilidade de escrever o código manualmente para o mesmo. Aqui, usaremos um *plug-in* chamado

jQuery Validation, e pode ser baixado no *site* [jQuery Validation](https://jqueryvalidation.org/) e clicando no link “Download”. Uma vez baixados os arquivos, e tendo o nosso formulário **HTML** pronto, e com a página *linkada* na biblioteca **jQuery**, basta *linkar* também no arquivo “jquery.validate.min.js”, ou claro, colocar seu conteúdo dentro do seu arquivo de comportamentos, e colocar ainda um código **jQuery** que atribui aos elementos a validação, como no exemplo abaixo:

```
<body>
  <form action="form.php" method="post" id="formulario">
    <fieldset>
      <legend>Dados pessoais</legend>
      <ul>
        <li>
          <label for="nome">Nome: </label>
          <input type="text" name="nome" id="nome">
        </li>
        <li>
          <label for="email">E-mail:</label>
          <input type="text" name="email" id="email">
        </li>
        <li>
          <label for="senha">Senha:</label>
          <input type="password" name="senha" id="senha">
        </li>
        <li>
          <label for="confirmar_senha">Confirme a senha:</label>
          <input type="password" name="confirmar_senha"
id="confirmar_senha">
        </li>
        <li>
          <input type="checkbox" name="termos" id="termos"
value="Concordo">
          <label for="termos">Concordo com os Termos de
Contrato</label>
        </li>
      </ul>
    </fieldset>
    <p>
      <input type="submit" value="Enviar">
    </p>
  </form>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"
integrity="sha256-/xUj+3OJU5yExlq6GSYGSgK7tPXikynS7ogEvDej/m4="
crossorigin="anonymous"></script>
  <script src="js/scripts.js"></script>
  <script>
    $(document).ready(function () {
      $('#formulario').validate({
        rules: {
          nome: {
            required: true,
            minlength: 3
```

```

    },
    email: {
      required: true,
      email: true
    },
    senha: {
      required: true
    },
    confirmar_senha: {
      required: true,
      equalTo: "#senha"
    },
    termos: "required"
  },
  messages: {
    nome: {
      required: "O campo nome é obrigatório.",
      minlength: "O campo nome deve conter no mínimo 3
caracteres."
    },
    email: {
      required: "O campo email é obrigatório.",
      email: "O campo email deve conter um email válido."
    },
    senha: {
      required: "O campo senha é obrigatório."
    },
    confirmar_senha: {
      required: "O campo confirmação de senha é obrigatório.",
      equalTo: "O campo confirmação de senha deve ser idêntico
ao campo senha."
    },
    termos: "Para se cadastrar você deve aceitar os termos de
contrato."
  }
});
});
</script>
</body>

```

Perceba que baseando-se na Id dada ao elemento no **HTML** nós atingimos o elemento no código **JS**. As mensagens podem ser trabalhadas da forma que preferir. Como pode ser observado na documentação do *plug-in* ([jQuery Validation - Documentação](#)), mais tipos de campos podem ser validados, como campos numéricos, data ou cartão de crédito. As mensagens de validação do formulário podem ser estilizadas via **CSS** através da classe `.error`.

15.3.2 Máscara

Outro comportamento conhecido para formulários é o uso de máscaras. Basicamente se baseia em definir qual é o padrão de preenchimento esperado para os campos, muito útil em campos como CEP, CPF, Telefone e afins. Utilizaremos o *plug-in* [jQuery Mask](#). Após baixado e inseridos os códigos **HTML** e **CSS** no projeto, é muito fácil ativar e configurar o *plug-in*, veja:

```
<body>
  <form action="form.php" method="post" id="formulario">
    <fieldset>
      <legend>Dados pessoais</legend>
      <ul>
        <li>
          <label for="telefone">Telefone: </label>
          <input type="tel" name="telefone" id="telefone">
        </li>
      </ul>
    </fieldset>
  </form>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"
integrity="sha256-/xUj+3OJU5yExlq6GSYGGShk7tPXikynS7ogEvDej/m4="
crossorigin="anonymous"></script>
  <script src="js/scripts.js"></script>
  <script>
    $(document).ready(function () {
      $("#telefone").mask("(00) 00000-0000");
    });
  </script>
</body>
```

15.4 Aplicações práticas do JavaScript

Agora que finalizamos o **JavaScript**, antes de continuarmos, vamos ver algumas das aplicações práticas mais comuns de vermos em *layouts* e que fazem uso de **JS**.

15.5 Topo Fixo

Um dos *scripts* mais comuns nos projetos é o que fixa o topo à medida que damos *scroll* na tela. Ele pode ser feito através de um *script* simples que verifica o posicionamento da barra de rolagem, veja:

```
// TOPO
$(window).scroll(function(){
  if(jQuery(this).scrollTop() > 100){
    $("header").addClass("topo-fixa");
  }else{
    $("header").removeClass("topo-fixa");
  }
});
```

Basicamente o *script* está dizendo que quando a barra de rolagem chegar a 100 pixels de rolagem, ele irá adicionar à *tag header* a classe **topo-fixa**. Agora basta definir um **position fixed** para esta classe.

15.6 Modal

Janelas modais é o nome dado para elementos se sobrepondo sobre toda a tela, com uma mensagem, um cadastro, algo do tipo. Em geral, demanda-se muito de **CSS** para sua construção, tendo um posicionamento e tamanho para exibir a janela naquele formato. Mas via **JS** iremos fazer o comportamento da opção fechar, para parar de exibir a janela modal.

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8">
    <title>Modal</title>
    <style>
      * {
        margin: 0;
        padding: 0;
        border: 0;
        list-style: none;
      }
      .modal-background {
        background: rgba(0, 0, 0, 0.7); /* Fundo transparente */
        width: 100%;
        height: 100%;
        position: fixed;
        z-index: 9;
      }
      .modal-content {
        background: #fff;
        width: 460px;
        height: 460px;
        position: absolute;
        z-index: 10;
        top: 50%;
        left: 50%;
```

```

        transform: translate(-50%, -50%); /* Centraliza a modal */
        padding: 20px;
    }
    .modal-close {
        position: absolute;
        right: -18px;
        top: -18px;
        z-index: 11;
        background: #000;
        color: #fff;
        border-radius: 50%;
        padding: 8px 10px;
        box-shadow: 0 0 5px #000;
        cursor: pointer;
    }
</style>
</head>
<body>
    <div class="modal">
        <div class="modal-background"></div>
        <div class="modal-content">
            <div class="modal-close">X</div>
            <h2>Título da Modal</h2>
            <p>Lorem ipsum, dolor sit amet consectetur adipisicing elit.
Quia, dolorem magni quod dolore doloribus, autem, officia nulla, a
tempore eum dolorum praesentium. Nulla at voluptatibus, culpa modi
nobis quibusdam! Aspernatur?</p>
            </div>
        </div>
        <script src="https://code.jquery.com/jquery-3.6.0.min.js"
integrity="sha256-/xUj+3OJU5yExlq6GSYGGShk7tPXikynS7ogEvDej/m4="
crossorigin="anonymous"></script>
        <script>
            $(document).ready(function () {
                $(".modal-close").click(function () {
                    $(".modal").hide();
                });
            });
        </script>
    </body>
</html>

```

Este exemplo possui várias aplicações práticas super interessantes de **CSS**, e o comportamento de fechamento da modal via **jQuery**.

15.7 Elemento deslizante

Muitas vezes em um *site* existe um elemento deslizante, como um menu para mobile, vejamos como fazer algo neste sentido com o uso de **CSS** e **jQuery**.

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8">
    <title>Modal</title>
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <style>
      * {
        margin: 0;
        padding: 0;
        border: 0;
        list-style: none;
      }
      nav {
        background: #900;
      }
      nav>ul {
        display: flex;
        justify-content: center;
      }
      nav a,
      nav p {
        color: #fff;
        display: block;
        padding: 10px 30px;
        text-decoration: none;
      }
      nav li:hover {
        background: #090;
      }
      nav li {
        position: relative;
      }
      nav ul ul {
        position: absolute;
        z-index: 2;
        top: 100%;
        left: 0;
        width: 100%;
        background: #CCC;
        text-align: center;
        display: none;
      }
      nav ul ul a {
        color: #000;
        padding: 10px;
      }
      nav li:hover ul {
```

```

    display: block;
  }
  /* MENU MOBILE */
  .bt-menu-mobile,
  .overlay-menu,
  nav p,
  .ver-todos,
  .close-menu {
    display: none;
  }
  .bt-menu-mobile {
    width: 40px;
    padding: 5px;
  }
  .bt-menu-mobile li {
    border: 2px solid #000;
    margin: 7px 0;
  }
  .overlay-menu {
    background: rgba(0, 0, 0, 0.7);
    width: 100%;
    height: 100%;
    position: fixed;
    z-index: 9;
    top: 0;
  }
  @media screen and (max-width:720px) {
    nav {
      position: fixed;
      left: -100%;
      top: 0;
      z-index: 10;
      width: 90%;
      background: #900;
      transition: .3s ease;
    }
    nav li {
      border-bottom: 1px solid #000;
    }
    .open-menu {
      left: 0;
      transition: .3s ease;
      height: 100%;
    }
    nav ul ul {
      position: relative;
      text-align: left;
    }
    nav ul ul li {
      padding-left: 30px;
    }
    .close-menu {
      position: absolute;
      top: 10px;

```

```

        right: 10px;
        background: #fff;
        border-radius: 50%;
        padding: 8px 10px;
        z-index: 11;
        line-height: 1;
    }
    nav li:hover ul,
    .item-mobile {
        display: none;
    }
    nav>ul,
    nav li ul.open-popup,
    .bt-menu-mobile,
    nav p,
    .ver-todos {
        display: block;
    }
}
</style>
</head>
<body>
    <div class="bt-menu-mobile">
        <ul>
            <li></li>
            <li></li>
            <li></li>
        </ul>
    </div>
    <div class="overlay-menu"></div>
    <div class="close-menu">X</div>
    <nav>
        <ul>
            <li><a href="#">Menu 1</a></li>
            <li><a href="#">Menu 2</a></li>
            <li>
                <a href="#" class="item-mobile">Menu 3</a>
                <p>Menu 3</p>
                <ul>
                    <li><a href="#">Submenu 1</a></li>
                    <li><a href="#">Submenu 2</a></li>
                    <li><a href="#">Submenu 3</a></li>
                    <li><a href="#">Submenu 4</a></li>
                    <li class="ver-todos"><a href="#">Ver todos Menu
3</a></li>
                </ul>
            </li>
            <li><a href="#">Menu 4</a></li>
            <li><a href="#">Menu 5</a></li>
        </ul>
    </nav>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"
integrity="sha256-/xUj+3OJU5yExlq6GSYGGShk7tPXikynS7ogEvDej/m4="
crossorigin="anonymous"></script>

```

```
<script>
  $(document).ready(function() {
    $(".bt-menu-mobile").click(function() {
      $(".nav").toggleClass("open-menu");
      $(".overlay-menu").show();
      $(".close-menu").show();
    });
    $(".overlay-menu, .close-menu").click(function() {
      $(".nav").removeClass("open-menu");
      $(".overlay-menu").hide();
      $(".close-menu").hide();
    });
    $(".nav p").click(function() {
      $(".nav p ~ ul").toggleClass("open-popup");
    });
  });
</script>
</body>
</html>
```



CAPÍTULO 16

BOOTSTRAP

Capítulo 16 - Bootstrap

Bootstrap é um *framework* de *front-end* que possui arquivos **CSS** e **JS** que facilitam o desenvolvimento de projetos. Um projeto quando linkado no **Bootstrap** possui classes padrão para desenvolvermos o projeto de forma muito mais prática e dinâmica, além de já possuir diversos comportamentos nativos para os *scripts* mais utilizados.

16.1 Principais vantagens

Um projeto que faz uso de **Bootstrap** tem algumas questões a serem levadas em conta. Primeiramente, é preciso considerar uma curva de aprendizado do desenvolvedor para se familiarizar com o *framework*, suas particularidades e afins. Também irá adicionar um pouco de peso ao projeto, por chamar arquivos com muitas linhas de código. No entanto, geralmente a troca acaba valendo a pena, pois uma vez familiarizados com o *framework*, o desenvolvedor ganha muita velocidade de desenvolvimento e na correção de possíveis *bugs*. Além disso, a responsividade é muito mais fácil, pois ele faz uso de classes padrão para o *grid-system* que veremos mais adiante.

O **Bootstrap** também possui alguns recursos muito úteis para o uso de ícones, de comportamentos comuns do **JavaScript**, além de já possuir um *reset* e uma padronização do projeto de forma nativa, não havendo a necessidade do desenvolvedor criar estes passos.

Confira a documentação e exemplos do uso do **Bootstrap** no [site oficial](#).

16.2 Inserindo Bootstrap no projeto

Toda a gama de serviços e vantagens oferecidas pelo uso do **Bootstrap** no seu projeto na verdade acaba se baseando muito no uso de arquivos **CSS** e **JS**, e estes arquivos permitem o uso das classes e recursos do *framework*. Claro que existem diversas versões e formas de trabalhar com ele, mas a mais comum acaba sendo este *link* nos arquivos.

O **Bootstrap** possui diversas versões, e elas mudam bastante de uma para outra. Aqui estaremos utilizando a versão 5, e todas as particularidades que ela nos trouxe.

Assim como no **jQuery**, podemos baixar os arquivos do **Bootstrap** e inseri-los no nosso projeto, ou *linkar* na versão já hospedada *online*. [O download pode ser feito aqui](#). No nosso caso optamos pela opção *linkada*, pois acaba sendo mais prática e não prejudica o projeto, então seguiremos neste formato. Veja um exemplo:

```

<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8">
    <title>Bootstrap</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/css/bootstrap
.min.css" rel="stylesheet"
integrity="sha384-+0n0xVW2eSR5OomGNYDnhzAbDsOXxcvSN1TPprVMTNDbiYZCxYb
OOl7+AMvyTG2x" crossorigin="anonymous">
    </head>
    <body>

      <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/js/bootstrap.b
undle.min.js"
integrity="sha384-gtEjrD/SeCtmISkJKNUaaKMoLD0//ElJl9smozuHV6z3Iehds+3
Ulb9Bn9Plx0x4" crossorigin="anonymous"></script>
    </body>
  </html>

```

Uma vez *linkado*, já é possível usar todos os recursos presentes no **Bootstrap** no projeto. Também existe um outro arquivo chamado **Popper**, que é uma outra biblioteca, que se *linkado* permitirá o uso de menus *pop-up* e janelas modais.

16.3 Grid System

Grid System é o nome dado para uma técnica criada a muito tempo para a estruturação de *layouts*. Basicamente se baseia em uma estrutura de colunas e espaçamento entre colunas, e vamos consumindo classes padrão no **CSS** para fazermos a estrutura do *site*. Veja uma imagem para ilustrar:

.col-xs-2	.col-xs-3	.col-xs-7
.col-xs-4	.col-xs-4	.col-xs-4
.col-xs-5	.col-xs-7	
.col-xs-6	.col-xs-6	
.col-xs-12		

A regra de ouro quando se trabalha com **Grid System** é que a soma das colunas precisa fechar em 12. Existem algumas classes padrão que podemos utilizar nas colunas para já adaptar o *layout* para responsivo, a tabela completa pode ser encontrada [aqui](#). As colunas devem estar sempre dentro de uma *tag* com a classe **row**, e essa por consequência deve estar sempre em uma *tag* com a classe **container**, para que o *site* fique centralizado, ou fora de uma *tag* com a classe *container* caso queira *full-screen*. Veja um exemplo:

```
<body>
  <div class="container">
    <div class="row">
      <div class="col-12 col-lg-3">

      </div>
      <div class="col-12 col-lg-9">

      </div>
    </div>
    <div class="row">
      <div class="col-6 col-lg-8">

      </div>
      <div class="col-6 col-lg-4">

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

16.4 Classes

Além das classes que permitirão a estruturação do *layout* através do **Grid System**, o **Bootstrap** também possui uma gama imensa de classes que permitem uma estilização

padrão de elementos como formulários, textos, imagens e outros elementos **HTML**. Basta definir a classe e a aparência desejada será renderizada. Uma lista completa de todas essas classes pode ser encontrada aqui: [Cheatsheet Bootstrap v5](#).

16.5 Ícones

O **Bootstrap** também possui toda uma biblioteca própria de ícones para podermos desenvolver o projeto. Para um melhor aproveitamento, seria interessante um alinhamento prévio com o *designer* do projeto para que ele use os mesmos ícones no desenvolvimento das telas. Para usar os ícones, basta copiar os códigos existentes no *site* do **Bootstrap** e colar no seu projeto. Uma lista completa dos ícones disponíveis pode ser vista aqui: [Bootstrap Icons](#).

16.6 Comportamentos

Ao *linkarmos* nosso projeto ao **CSS** e **JS** do **Bootstrap**, ele também nos permite uma gama imensa de comportamentos comuns em *sites* para podermos aplicar ao nosso documento. Vejamos alguns mais utilizados:

16.6.1 Accordion

O efeito de **Accordion** se baseia em clicarmos em um elemento e assim ele abrir o seu conteúdo, bastante utilizado em páginas de perguntas e respostas ou de longos textos em tópicos. Vejamos um exemplo:

```
<body>

<div class="accordion" id="accordionExample">

  <div class="accordion-item">

    <h2 class="accordion-header" id="headingOne">

      <button class="accordion-button" type="button"
data-bs-toggle="collapse" data-bs-target="#collapseOne"
aria-expanded="true" aria-controls="collapseOne">

        Accordion Item #1

      </button>
```

```

    </h2>

    <div id="collapseOne" class="accordion-collapse collapse show"
aria-labelledby="headingOne" data-bs-parent="#accordionExample">

        <div class="accordion-body">

            <strong>This is the first item's accordion body.</strong> It is
shown by default, until the collapse plugin adds the appropriate classes
that we use to style each element. These classes control the overall
appearance, as well as the showing and hiding via CSS transitions. You can
modify any of this with custom CSS or overriding our default variables.
It's also worth noting that just about any HTML can go within the
<code>.accordion-body</code>, though the transition does limit overflow.

        </div>

    </div>

</div>

<div class="accordion-item">

    <h2 class="accordion-header" id="headingTwo">

        <button class="accordion-button collapsed" type="button"
data-bs-toggle="collapse" data-bs-target="#collapseTwo"
aria-expanded="false" aria-controls="collapseTwo">

            Accordion Item #2

        </button>

    </h2>

    <div id="collapseTwo" class="accordion-collapse collapse"
aria-labelledby="headingTwo" data-bs-parent="#accordionExample">

        <div class="accordion-body">

            <strong>This is the second item's accordion
body.</strong> It is hidden by default, until the collapse plugin adds the
appropriate classes that we use to style each element. These classes
control the overall appearance, as well as the showing and hiding via CSS
transitions. You can modify any of this with custom CSS or overriding our
default variables. It's also worth noting that just about any HTML can go
within the <code>.accordion-body</code>, though the transition does limit
overflow.

        </div>

    </div>

</div>

<div class="accordion-item">

```

```

<h2 class="accordion-header" id="headingThree">
    <button class="accordion-button collapsed" type="button"
data-bs-toggle="collapse" data-bs-target="#collapseThree"
aria-expanded="false" aria-controls="collapseThree">
        Accordion Item #3
    </button>
</h2>

<div id="collapseThree" class="accordion-collapse collapse"
aria-labelledby="headingThree" data-bs-parent="#accordionExample">
    <div class="accordion-body">
        <strong>This is the third item's accordion body.</strong> It
is hidden by default, until the collapse plugin adds the appropriate
classes that we use to style each element. These classes control the
overall appearance, as well as the showing and hiding via CSS transitions.
You can modify any of this with custom CSS or overriding our default
variables. It's also worth noting that just about any HTML can go within
the <code>.accordion-body</code>, though the transition does limit
overflow.
    </div>
</div>
</div>
</div>
</body>

```

As variações disponíveis nativamente para o efeito de **Accordion** do **Bootstrap** podem ser encontradas aqui: [Accordion - Bootstrap](#)

16.6.2 Tabs

Outro efeito comum em projetos é o **Tabs** (Abas), muito utilizado para diversas descrições de um mesmo elemento, como um produto, ou em páginas de apresentação de algum conteúdo, e é possível fazer este efeito com o **Bootstrap**. Vamos ver um exemplo:

```

<body>

<ul class="nav nav-tabs" id="myTab" role="tablist">

```

```

    <li class="nav-item" role="presentation">
      <button class="nav-link active" id="home-tab" data-bs-toggle="tab"
data-bs-target="#home" type="button" role="tab" aria-controls="home"
aria-selected="true">Home</button>
    </li>

    <li class="nav-item" role="presentation">
      <button class="nav-link" id="profile-tab" data-bs-toggle="tab"
data-bs-target="#profile" type="button" role="tab" aria-controls="profile"
aria-selected="false">Profile</button>
    </li>

    <li class="nav-item" role="presentation">
      <button class="nav-link" id="contact-tab" data-bs-toggle="tab"
data-bs-target="#contact" type="button" role="tab" aria-controls="contact"
aria-selected="false">Contact</button>
    </li>
  </ul>

  <div class="tab-content" id="myTabContent">
    <div class="tab-pane fade show active" id="home" role="tabpanel"
aria-labelledby="home-tab">
      <p>lipsum Commodo anim cillum excepteur anim dolor dolor mollit
occaecat velit officia tempor aliqua commodo ex. Deserunt officia
exercitation sit incididunt culpa quis nostrud fugiat eu pariatur veniam
deserunt culpa. Amet laborum occaecat dolore mollit magna dolor elit.</p>
    </div>

    <div class="tab-pane fade" id="profile" role="tabpanel"
aria-labelledby="profile-tab">
      <p>Exercitation fugiat cupidatat sit nisi sint dolor incididunt
enim. Qui ex Lorem do occaecat ex eu. Enim non non et incididunt sit
consectetur enim excepteur velit do duis eu amet. Dolor consequat qui
culpa dolore magna. Dolore magna irure dolore in aute incididunt esse
irure. Dolor nulla consectetur consectetur officia commodo excepteur
consectetur aliqua qui non. Proident magna nisi nulla in veniam mollit
cupidatat aliqua in voluptate ipsum ullamco tempor.</p>
    </div>

    <div class="tab-pane fade" id="contact" role="tabpanel"
aria-labelledby="contact-tab">

```

```
<p>Pariatur aliquip officia occaecat sunt labore aliqua proident  
nostrud ipsum consequat anim. Voluptate incididunt incididunt consequat  
quis excepteur tempor id non ea eiusmod. Sunt fugiat dolor quis duis  
consectetur. Dolor proident tempor cupidatat ipsum deserunt ad mollit  
tempor cillum deserunt aliquip occaecat nostrud sunt. Eiusmod laborum qui  
officia commodo. Minim ut cillum ullamco incididunt id. Eu laborum Lorem  
cupidatat laborum deserunt ex nulla.</p>  
  
</div>  
  
</div>  
  
</body>
```

Uma documentação completa com todas as variações possíveis para **Tabs** pode ser encontrada aqui: [Navs and tabs - Bootstrap](#).



CAPÍTULO 17

FLUXOS DE **DESENVOLVIMENTO**

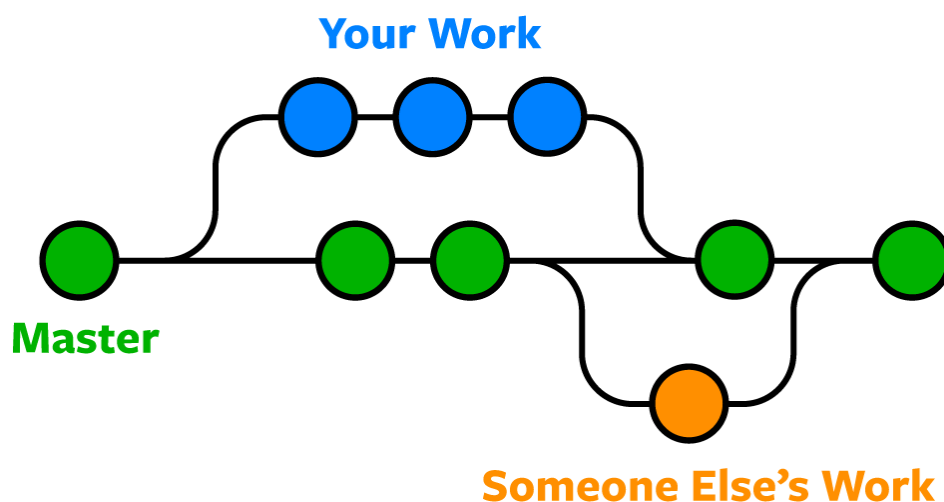
Capítulo 17 - Fluxos de desenvolvimento

Existem algumas práticas que são comuns de serem encontradas quando começamos a trabalhar em empresas de tecnologia. Seria interessante já termos alguma familiaridade com as mais utilizadas, vejamos:

17.1 - Git e GitHub

Git é uma tecnologia que permite o versionamento de projetos. Ele se baseia em uma “linha-mestra”, com a versão atual do projeto, chamada de **master**, e esta **master** pode ser clonada para um repositório apartado, próprio daquele desenvolvedor, chamada de **branch**. Desta forma o desenvolvedor consegue trabalhar no projeto sem de forma alguma afetar a versão original.

Uma vez que o desenvolvedor finalizou uma etapa do projeto, ele pode solicitar o envio desta versão para ser revisada por algum outro desenvolvedor, este envio é chamado de **commit**. Caso haja conflitos entre a **master** e a **branch**, o desenvolvedor que trabalhou nesta **branch**, nestas alterações do código original, deve corrigir os conflitos. Somente quando temos a certeza que aceitar estas alterações não irá afetar negativamente o projeto da **master**, é possível fazer o **merge**, que é mesclar na **master** os ajustes feitos na **branch**. Este ilustração pode auxiliar a compreensão:



Trabalhar com **Git** é uma boa prática pois permite ter sempre *backups* do projeto (versões), além de garantir que testes podem ser realizados antes que este código se torne o código principal. Todas as grandes empresas de tecnologia seguem este fluxo.

Tudo isso acontece através de *softwares* que você pode instalar no seu computador, como **Git**, **Source-tree** e outros, para facilitar o entendimento das versões.

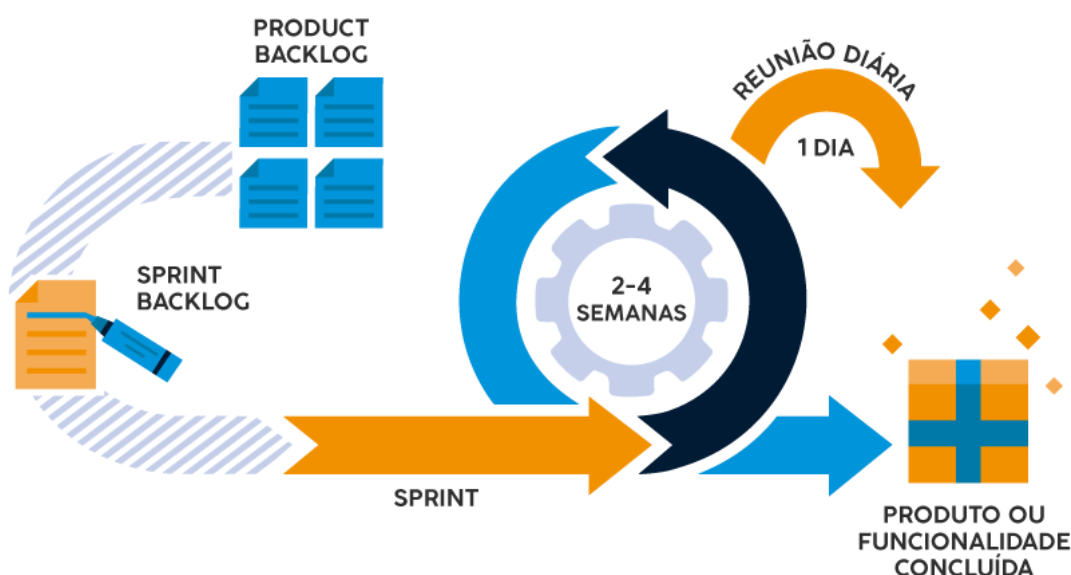
Github é uma rede social para desenvolvedores. Lá você pode publicar um trabalho que você realizou, e outros desenvolvedores poderão ver o seu projeto, inclusive

baixar para suas próprias máquinas, realizarem ajustes e enviar novamente para você, assim criando uma comunidade que evolui junto. Através do **Github** diversas ferramentas incríveis que utilizamos hoje foram criadas.

17.2 Scrum e Kanban

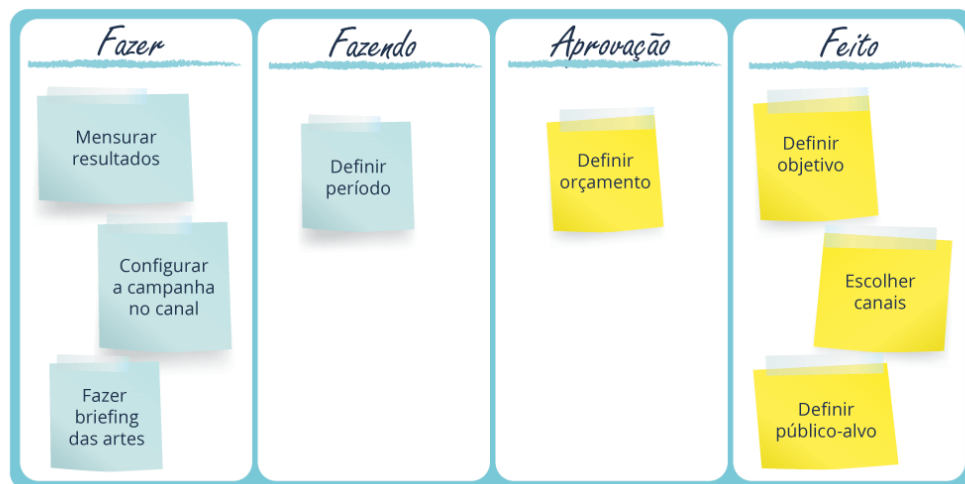
Scrum e **Kanban** são técnicas para desenvolvimento ágil e gestão de projetos. Os desenvolvedores utilizam para facilitar a visão do projeto e acelerar o desenvolvimento do mesmo.

Scrum é uma metodologia muito utilizada nas empresas de desenvolvimento para garantir pequenas entregas e perfeito alinhamento do time. Em resumo, é feita uma contratação de desenvolvimentos que o **PO** (*Project Owner*) gostaria que fossem realizados em um determinado período de tempo. Por exemplo, digamos que se deseja que algumas páginas do projeto sejam finalizadas em 3 semanas. O PO documenta tudo o que ele deseja como critérios de aceitação para o projeto ser considerado como entregue. Na sequência a equipe trabalha por 3 semanas fazendo reuniões diárias (*Dailys*), para garantir o alinhamento de todos, definindo o que cada fez, está fazendo e irá fazer. Por fim é feita a entrega do projeto que precisa atender à todos os critérios de aceite. Um estudo mais completo sobre o assunto pode ser feito através deste livro: **SCRUM: a arte de fazer o dobro do trabalho na metade do tempo**. A imagem abaixo ilustra um pouco sobre as cerimônias envolvendo **Scrum**:



Já o **Kanban** é uma outra técnica de gestão que facilita a visualização e entendimento do projeto ou de todos os projetos que os profissionais estão atuando.

Geralmente é feita em um quadro, quando fisicamente, ou em um *software*, como o **Trello**, quando virtualmente. Geralmente se divide em algumas colunas como “A fazer, fazendo e feito”, e se colocam os *cards* ou post-its de tudo o que se tem evoluído. Veja esta ilustração para facilitar o entendimento:



17.3 Gestão do tempo

Para tentar garantir que o tempo de todos os profissionais do time estão sendo amplamente aproveitados, utiliza-se de algumas técnicas de gestão do tempo, como por exemplo o **GTD** (*Getting Things Done*) ou o **Pomodoro**. Sobre **GTD** é possível ter mais detalhes no livro **A arte de fazer acontecer**, mas se baseia em organizar suas tarefas de forma que se possa atender imediatamente ou colocar em uma fila lógica de importância.

Já o **Pomodoro** é uma técnica onde se pega uma faixa de tempo específica, como 25 minutos por exemplo, e se foca totalmente no trabalho por este tempo, e depois se faz uma pausa, indo na sequência para o próximo “pomodoro”. Existem extensões do **Chrome** que facilitam o trabalho e o controle destes intervalos de tempo.



CAPÍTULO 18

LINKS E **TABELAS** COMPLEMENTARES

Capítulo 18 - Links e tabelas complementares

18.1 Tags HTML

Lista das *tags* **HTML** em [W3Schools - Tags](#)

18.2 Lista de novas tags HTML5

Segue uma lista baseada nos novos elementos **HTML5**.

Tag	Função
<article>	Cria um campo para artigo.
<aside>	Conteúdo relacionado ao artigo ou para criação de <i>sidebar</i> .
<audio>	Áudio em um documento HTML .
<bdi>	Parte de um texto que pode ser formatado em diferentes direções.
<canvas>	Criação de gráficos.
<command>	Comando que o usuário pode invocar.
<datalist>	Autocompletar em formulários como exemplos para o preenchimento do campo. Atualmente funciona apenas no Ópera e no Mozilla Firefox .
<datatemplate>	Template de dados.
<details>	Informações adicionais que o usuário pode visualizar ou esconder.
<device>	Permite ao usuário dar o acesso à página para um dispositivo.
<embed>	Aplicação externa com conteúdo interativo.
<figcaption>	Define uma legenda para um elemento <figure>.
<figure>	Ilustrações, imagens, fotos, associadas juntamente com alguma legenda.
<footer>	Cria uma seção na página que pode ser utilizada para criação do Rodapé.
<header>	Cria uma seção na página que pode ser utilizada para criação do Topo.
<keygen>	Chave de acesso.
<main>	Define a área de conteúdo do site.
<mark>	Texto destacado para fins de referência.

<meter>	Medição dentro de um intervalo predefinido.
<nav>	Região para links para navegação.
<output>	Resultado de um cálculo.
<progress>	Progresso de uma tarefa como a execução de um script em Java.
<rt>	Componente do texto em uma anotação.
<section>	Seção para um artigo.
<source>	Exibe múltiplos recursos multimídia.
<time>	Define uma região para conter uma unidade de tempo.
<track>	Legenda para elementos multimídia.
<video>	Vídeo em um documento html.
<wbr>	Possível quebra de linha.

18.3 Atributos Globais HTML5

Atributos globais são atributos que podem ser inseridos em qualquer *tag*. Com **HTML5** muitos atributos que eram específicos se tornaram globais e outros foram incorporados. Veja a lista:

Atributo	Função
accesskey	Especifica uma tecla de atalho para dar foco ao elemento.
class	Atribui um nome de classe para ser estilizado por folhas de estilo.
contenteditable	Especifica se o elemento é editável.
contextmenu	Atribui um menu de contexto para aparecer quando o usuário utilizar o botão auxiliar do mouse.
dir	Direção do texto.
draggable	Especifica se o elemento é passível de ser arrastado.
dropzone	Especifica se o elemento arrastado será copiado, movido ou linkado.
hidden	Esconde o elemento.
id	Atribui uma identificação ao elemento para ser estilizado por folhas de estilo.
lang	Especifica o idioma.
spellcheck	Atribui um corretor ortográfico.
style	Para estilizações <i>inline</i> .

tabindex	Especifica a ordem de acesso pela tecla tab.
title	Texto ao posicionar o ponteiro do mouse sobre o elemento.

18.4 Lista de tags aposentadas (*deprecated*) no HTML5

- <acronym>
- <applet>
- <basefont>
- <big>
- <center>
- <dir>
-
- <frame>
- <frameset>
- <isindex>
- <noframes>
- <s>
- <strike>
- <tt>

Também alguns atributos de *tags* foram aposentados, como:

- align para caption, iframe, img, input, object, legend, table, hr, div, h1, h2, h3, h4, h5, h6, p, col, colgroup, tbody, td, tfoot, th, thead e tr;
- abbr e scope para td;
- alink, link, text, vlink, background para body;
- border para object;
- bgcolor para table, tr, td, th and body;
- cellpadding e cellspacing para table;
- frameborder e scrolling para iframe;
- height para td e th;
- type para li e ul;
- summary para table;
- valign para col, colgroup, tbody, td, tfoot, th, thead e tr;

- width para hr, table, td, th, col, colgroup and pre.

18.5 Lista de Seletores CSS

Seletor	Função
*	Seletor universal: qualquer elemento.
.class	Seletor de classe.
#id	Seletor de ID.
e	Seletor de tipo de elemento: seleciona qualquer elemento <e>.
e f	Seletor contextual: seleciona qualquer elemento <f> que estiver contido num elemento <e>.
e > f	Seletor de elementos-filho: seleciona qualquer elemento <f> descendente direto de um elemento <e>.
e + f	Seletor adjacente: seleciona o elemento <f> que estiver imediatamente após um elemento <e>.
e , f	Agrupamento de seletores: Seleciona os elementos <e> e <f> do agrupamento.
e [atrib]	Seletor de Atributo: seleciona o elemento <e> que contenha o atributo indicado.
e [atrib="valor"]	Seletor de Atributo: seleciona o elemento <e> com o mesmo atributo e “valor”.
e:first-child	Pseudo-classe primeiro filho: seleciona o primeiro elemento <e> descendente do elemento-pai.
a:link	Pseudo-classe link: aplica-se ao elemento <a> com hiperlinks ou âncoras ainda não visitados.
a:visited	Pseudo-classe visited: aplica-se ao elemento <a> com hiperlinks ou âncoras já visitados.
e:active	Pseudo-classe active: aplica-se ao elemento <e> quando este for ativado pelo usuário.
e:hover	Pseudo-classe hover: aplica-se ao elemento <e> quando o cursor estiver sobre ele, mas sem ativá-lo.
e:focus	Pseudo-classe focus: aplica-se ao elemento <e> quando o foco estiver posicionado nele.
e:first-line	Pseudo-elemento first-line: aplica-se à primeira linha do elemento <e>.

e:first-letter	Pseudo-elemento first-letter: aplica-se à primeira letra do elemento <e>.
e:before	Pseudo-elemento before: aplica conteúdo especificado em posição anterior ao elemento <e>.
e:after	Pseudo-elemento after: aplica conteúdo especificado em posição posterior ao elemento <e>.
e:last-child	Pseudo-classe último filho: seleciona o último elemento <e> descendente do elemento-pai.

18.6 Lista de propriedades CSS

Propriedade	Valores
background	Color image repeat attachment position.
background-attachment	Fixed ou scroll.
background-image	url(nomedaimagem.jpg).
background-color	Código hexadecimal ou nome da cor.
background-position	Valor numérico seguido da unidade de medida. Primeiro valor para esquerda, segundo para topo.
background-repeat	No-repeat, repeat, repeat-x ou repeat-y.
border	width style color.
border-color	Código hexadecimal ou nome da cor.
border-style	None, dotted, dashed, solid, double, groove, ridge, inset ou outset.
border-width	Valor numérico seguido da unidade de medida.
border-bottom	width style color.
border-bottom-color	Código hexadecimal ou nome da cor.
border-bottom-style	None, dotted, dashed, solid, double, groove, ridge, inset ou outset.
border-bottom-width	Valor numérico seguido da unidade de medida.
border-left	width style color.
border-left-color	Código hexadecimal ou nome da cor.
border-left-style	None, dotted, dashed, solid, double, groove, ridge, inset ou outset.
border-left-width	Valor numérico seguido da unidade de medida.

border-right	width style color.
border-right-color	Código hexadecimal ou nome da cor.
border-right-style	None, dotted, dashed, solid, double, groove, ridge, inset ou outset.
border-right-width	Valor numérico seguido da unidade de medida.
border-top	width style color.
border-top-color	Código hexadecimal ou nome da cor.
border-top-style	None, dotted, dashed, solid, double, groove, ridge, inset ou outset.
border-top-width	Valor numérico seguido da unidade de medida.
clear	None, left, right ou both.
color	Código hexadecimal ou nome da cor.
cursor	Default, pointer, crosshair, e-resize, help, move, ne-resize, n-resize, nw-resize, se-resize, s-resize, sw-resize, text, wait ou w-resize.
display	Block, inline, none, inline-block, inline-table, compact, list-item ou table.
font	Style variant weight size family.
font-family	Família de fontes.
font-size	Valor numérico seguido da unidade de medida.
font-style	Normal, italic ou oblique.
font-variant	Normal ou small-caps.
font-weight	Normal, bold, bolder, lighter ou valor numérico.
height	Valor numérico seguido da unidade de medida.
left	Valor numérico seguido da unidade de medida.
letter-spacing	Valor numérico seguido da unidade de medida.
line-height	Valor numérico seguido da unidade de medida.
list-style	Position; imagem ou position; type.
list-style-image	None ou url(nomedaimagem.jpg).
list-style-position	Outside ou inside.
list-style-type	None, disc, circle, square, decimal, lower-roman, upper-roman, lower-alpha ou upper-alpha.
margin	Top right bottom left.
margin-left	Valor numérico seguido da unidade de medida.

margin-right	Valor numérico seguido da unidade de medida.
margin-bottom	Valor numérico seguido da unidade de medida.
margin-top	Valor numérico seguido da unidade de medida.
overflow	Visible, hidden, scroll ou auto.
padding	Valor numérico seguido da unidade de medida.
padding-bottom	Valor numérico seguido da unidade de medida.
padding-left	Valor numérico seguido da unidade de medida.
padding-right	Valor numérico seguido da unidade de medida.
padding-top	Valor numérico seguido da unidade de medida.
position	Absolute, relative, fixed ou static.
text-align	Left, right, center ou justify.
text-decoration	None, underline, overline, line-through ou blink.
text-indent	Valor numérico seguido da unidade de medida.
text-transform	None, capitalize, uppercase ou lowercase.
top	Valor numérico seguido da unidade de medida.
vertical-align	Baseline, bottom, middle, sub, super, text-bottom, text-top ou top.
visibility	Hidden ou visible.
width	Valor numérico seguido da unidade de medida.
Word-spacing	Valor numérico seguido da unidade de medida.
z-index	Valor numérico.

18.7 Lista das novas propriedades CSS3

Propriedade	Valores
backface-visibility	Hidden.
background-origin	Border-box, padding-box ou content-box.
background-clip	Border-box, padding-box ou content-box.
background-size	Valor numérico seguido da unidade de medida como primeiro valor para largura e Valor numérico seguido da unidade de medida para valor da altura.
border-color	Código hexadecimal ou nome da cor.

border-image	url(imagem.jpg) para primeiro valor, Valor numérico seguido da unidade de medida para segundo e terceiro valor e valor round para quarto valor.
border-radius	Valor numérico seguido da unidade de medida.
box-sizing	Content-box ou border-box.
box-shadow	Valor numérico seguido da unidade de medida para os três primeiros valores e no quarto valor código hexadecimal ou nome da cor.
column-count	Valor numérico.
column-gap	Valor numérico seguido da unidade de medida.
column-rule	No primeiro valor definimos um valor numérico seguido da unidade de medida, em seguida um estilo, que pode ser None, dotted, dashed, solid, double, groove, ridge, inset ou outset e por fim o Código hexadecimal ou nome da cor.
opacity	Valor numérico entre zero e 1.
outline-offset	Valor numérico seguido da unidade de medida.
perspective	Valor numérico.
perspective-origin	Um primeiro valor numérico seguido da unidade de medida para o eixo x e um segundo Valor numérico seguido da unidade de medida para o eixo y.
resize	None, both, horizontal ou vertical.
text-shadow	Os primeiros dois valores são valores numéricos seguidos da unidade de medida para deslocamento horizontal e vertical respectivamente, um terceiro valor para enevoamento e o quarto valor como código hexadecimal ou nome da cor.
text-overflow	Clip, ellipsis ou string.
transform	None, matrix, matrix3d, translate, translate3d, translateX, translateY, translateZ, scale, scale3d, scaleX, scaleY, scaleZ, rotate, rotate3d, rotateX, rotateY, rotateZ, skew, skewX, skewY ou perspective.
transition	Primeiro valor com all ou a definição do que animar, segundo valor em tempo e terceiro valor como ease, ease-in, ease-out, ease-in-out, linear ou cubic-bezier.
word-wrap	Normal ou break-word.
@keyframes	Para definições de animações.
@font-face	Para definição de fontes externas.

18.8 Tabela de fontes-padrão

Como vimos, não poderemos utilizar todas as fontes que quisermos no nosso código **HTML** e **CSS**. Por questão de acessibilidade utilizamos somente fontes-padrão, que são fontes que temos a certeza de que existem em todos os sistemas operacionais modernos (a não ser que se utilize `@font-face` do **CSS3** ou a **API** de *fonts* do **Google**). No caso de **CSS** utilizamos famílias de fontes, para caso a primeira fonte não exista no computador do usuário, ela seja substituída pela seguinte e assim sucessivamente para exibir a informação. Veja quais são:

Fontes-padrão	Famílias de fontes
Arial	Arial, Helvetica, sans-serif
Arial Black	Arial Black, Gadget, sans-serif
Comic Sans MS	Comic Sans MS, cursive
Courier New	Courier New, Courier, monospace
Georgia	Georgia, Times New Roman, Times, serif
Lucida Console	Lucida Console, Monaco, monospace
Lucida Sans Unicode	Lucida Sans Unicode, Lucida Grande, sans-serif
MS Serif	MS Serif, New York, serif
Palotino Linotype	Palotino Linotype, Book Antiqua, Palotino, serif
Tahoma	Tahoma, Geneva, sans-serif
Times New Roman	Times New Roman, Times, serif
Trebuchet MS	Trebuchet MS, Arial, Helvetica, sans-serif
Verdana	Verdana, Geneva, sans-serif

No caso das **CSS**, devemos colocar uma fonte que tenha mais de um nome em sua descrição entre aspas. Exemplo: “Trebuchet MS”, Arial, Helvetica, sans-serif.

18.9 Tabela de caracteres

Em um documento **HTML** podemos escrever caracteres com acentos, cedilha e caracteres especiais, mas pode acontecer de um usuário que utilize um idioma padrão em seu sistema operacional que não contenha esses caracteres. Neste caso, os caracteres não serão exibidos, ou serão exibidos de forma deturpada, como um caractere inválido. Para evitar este tipo de problema, por questão de acessibilidade devemos procurar digitar conteúdos com caracteres diferentes no formato de códigos que levam a ele e são interpretados pelos navegadores em qualquer idioma. São eles:

Carac.	Código	Carac.	Código	Carac.	Código	Carac.	Código
Á	Á	É	É	Ô	Ô	®	®
á	á	é	é	ô	ô	©	©
Â	Â	Ê	Ê	Õ	Õ	Ñ	Ñ
â	â	ê	ê	õ	õ	ñ	ñ
À	À	Í	Í	Ú	Ú	Ë	Ë
à	à	í	í	ú	ú	ë	ë
Ã	Ã	Ó	Ó	ç	ç	Ü	Ü
ã	ã	ó	ó	&	&	ü	ü

18.10 Métodos jQuery

Veja a tabela dos Métodos **jQuery** mais utilizados:

Método	Descrição
bind()	Liga eventos em elementos.
blur()	Atribui/Inicia o evento blur.
change()	Atribui/Inicia o evento change.
click()	Atribui/Inicia o evento click.
dblclick()	Atribui/Inicia o evento duplo click.
focus()	Atribui/Inicia o evento focus.
focusin()	Atribui um evento a um evento focusin.
focusout()	Atribui um evento a um evento focusout.
hover()	Liga dois eventos a um evento hover.
mousedown()	Atribui/Inicia o evento mousedown.
mouseenter()	Atribui/Inicia o evento mouseenter.
mouseleave()	Atribui/Inicia o evento mouseleave.
mousemove()	Atribui/Inicia o evento mousemove.
mouseout()	Atribui/Inicia o evento mouseout.

mouseover()	Atribui/Inicia o evento mouseover.
mouseup()	Atribui/Inicia o evento mouseup.
trigger()	Inicia todos os eventos ligados aos elementos selecionados.
scroll()	Atribui/Inicia o evento scroll.
unbind()	Remove os eventos adicionados dos elementos selecionados.
select()	Atribui/Inicia o evento select.
toggle()	Atribui duas ou mais funções a alternar entre um clique e outro (removido na versão 1.9).
ready()	Atribui executar a função quando o DOM for totalmente baixado.
submit()	Atribui/Inicia o evento submit.
resize()	Atribui/Inicia o evento resize.

Dica: Para uma lista completa e exemplos de uso de cada um dos métodos, acesse [W3Schools - jQuery Events](https://www.w3schools.com/jquery/jquery_events.asp).

18.11 Efeitos jQuery

Veja a tabela dos efeitos **jQuery**:

Efeito	Descrição
clearQueue()	Remove todas as filas de funções restantes dos elementos selecionados.
delay()	Define uma espera para todas as filas de funções nos elementos selecionados.
dequeue()	Remove a próxima função da fila, e então executa a função.
fadeIn()	Efeito de Fade in nos elementos selecionados.
fadeOut()	Efeito de Fade out nos elementos selecionados.
fadeTo()	Efeito de Fade in/out nos elementos selecionados, atribuindo opacidade.
fadeToggle()	Alterna entre efeito de Fade in/out nos elementos selecionados.
finish()	Encerra, remove e completa todas as filas de animações nos elementos selecionados.

hide()	Esconde elementos selecionados.
queue()	Mostra a fila de funções nos elementos selecionados.
show()	Mostra elementos selecionados.
slideDown()	Efeito slide-down (mostrar) nos elementos selecionados.
slideToggle()	Alterna entre efeitos slide-down e slide-up nos elementos selecionados.
slideUp()	Efeito slide-down (esconder) nos elementos selecionados.
stop()	Para a animação atual nos elementos selecionados.
toggle()	Alterna entre efeitos hide e show nos elementos selecionados.



CAPÍTULO 19

CONCLUSÃO

Capítulo 19 - Leitura complementar

- No que diz respeito à semântica e padrões *web* é sempre bom conferir o site nacional da [W3C](#);
- A própria **W3C** oferece um [portal na web](#) para o aprendizado de novos efeitos e suportes a **HTML**, **CSS**, **HTML5** e **CSS3**;
- Para um aprendizado didático e aprofundado, consulte o site da [W3Schools](#).

19.1 Referências para consulta

- No que diz respeito as **CSS**, o site [Maujor](#) é a maior referência nacional atual;
- Lista com todas as principais *tags HTML* em [Codigo Fonte - Principais tags de HTML](#);
- Guia de referência oficial da **W3C** a respeito de [CSS2](#);
- Para dicas sobre como melhorar o desempenho do seu projeto, o site [BrowserDiet](#) é a melhor referência atual;
- Consulte a documentação oficial da [jQuery](#).

19.2 Mais tabelas

- Confira ainda a [tabela dos métodos HTML / CSS](#) para **jQuery**;
- Confira a tabela de [métodos de travessia jQuery](#).

19.3 Glossário do Dev

Alguns termos são comuns de serem utilizados na área de desenvolvimento *web*. Para que ninguém fique perdido em uma *call* ou reunião, vamos aprender um pouco deste vocabulário próprio.

Termo	Significado
Bug	Erro em um <i>script</i> que gera uma incoerência no código.
Issue	Semelhante ao <i>bug</i> , mas geralmente se remete a erros um pouco maiores.
Hotfix	Uma correção rápida de um erro no <i>site</i> feita de forma pontual.
SLA	<i>Service Level Agreement</i> , garante a qualidade do serviço prestado. Geralmente a sigla é utilizada em chamados e tem a ver com a urgência do mesmo e seu prazo de resolução.

SSL	Certificado de segurança do site, fazendo a <i>URL</i> ter https:// .
Feature	O lançamento de uma nova implementação.
Versões	Versões de um <i>software</i> são geralmente controladas com três níveis numéricos, por exemplo 1.0.17, onde o primeiro número é chamado de <i>major</i> , o segundo <i>minor</i> e o terceiro <i>patch</i> .
CS e CX	Customer Success (CS) é um termo geralmente utilizado para o atendimento ao cliente focado em seu sucesso. Customer Experience (CX) é um termo geralmente utilizado para falar sobre a experiência do cliente ao navegar no <i>site</i> .
PJ	Pessoa Jurídica, geralmente utilizado para se referir a forma de contrato que o desenvolvedor tem com a empresa.
MEI	Micro Empreendedor Individual, é uma sigla bastante comum na nossa área, onde o desenvolvedor presta serviços para a empresa, mas não sob o regime CLT, e sim como uma empresa contratada.
B2B, B2C	Business to business (B2B) é uma empresa que atende apenas outras empresas. Business to Customer (B2C) é uma empresa que atende ao cliente final.
Wave	Uma onda geralmente é uma etapa a ser entregue do projeto.
UX e UI	User Experience (UX) se refere à experiência do cliente ao navegar pelo <i>site</i> . User Interface (UI) se refere à estrutura de páginas que o cliente navega.
QA	Quality Assurance . Geralmente o termo é utilizado para a validação da qualidade e fidelidade do projeto.

19.4 Referências bibliográficas:

- Samy Silva, Maurício; **HTML5 A Linguagem de Marcação que Revolucionou a Web – Primeira Edição**; Novatec Editora, 2011;
- Samy Silva, Maurício; **CSS3 : Desenvolva aplicações web profissionais com uso dos poderosos recursos de estilização das CSS3 – Primeira Edição**; Novatec Editora, 2012;
- Samy Silva, Maurício; **JavaScript: guia do programador – Primeira Edição**; Novatec Editora, 2010;
- Samy Silva, Maurício; **jQuery: a biblioteca do programador JavaScript – Segunda Edição**; Novatec Editora, 2010.