Web front-end

LET'S CODE
LET'S CODE
LET'S CODE
LET'S CODE
LET'S CODE
LET'S CODE
LET'S CODE
LET'S CODE
LET'S CODE
LET'S CODE
LET'S CODE
LET'S CODE
LET'S CODE
LET'S CODE
LET'S CODE
LET'S CODE
LET'S CODE
LET'S CODE
LET'S CODE
LET'S CODE
LET'S CODE

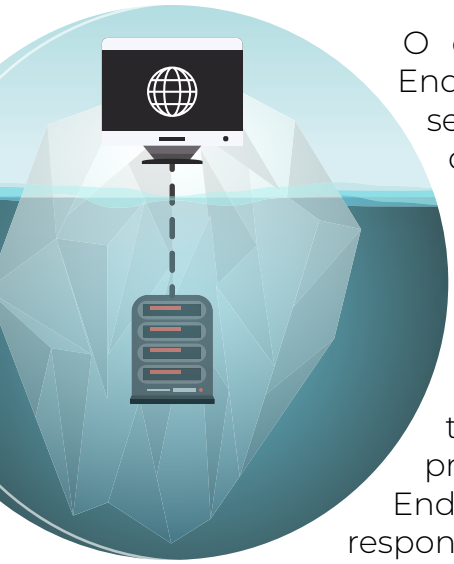
SUMÁRIO

CAPÍTULO

01

CONCEITO

1.1 Introdução ao conceito de desenvolvedor front-end



O conceito de Front-End, ou interface frontal, se refere a todo o desenvolvimento realizado na parte visual do site, em computação. Tudo aquilo que você vê ao acessar um site faz parte do trabalho de um profissional de Front-End. Este pode ser responsável pela definição do *layout*; pela disposição das informações, cores e conteúdo na tela; e/ou pela codificação da parte técnica, para que o site funcione corretamente.

O profissional envolvido neste tipo trabalho costuma possuir vários nomes no mercado, podendo ser chamado de **Desenvolvedor Front-End, Desenvolvedor Web, Desenvolvedor de Interface**, entre outras denominações. É importante destacar que algumas das tarefas do desenvolvimento de um site podem ser distribuídas para outros profissionais também, que possuem expertises específicas para a realização de certas funções. Por exemplo, a definição do design de um site pode não ser responsabilidade do programador Front-End, um designer gráfico pode assumir este papel e realizar este trabalho utilizando *softwares* de prototipagem. Assim o designer pode mostrar como o layout deve ser desenvolvido, qual o tipo de letra (fontes) que deve ser utilizado, além de cores, espaçamentos, imagens, e toda a combinação destes itens, já com sua disposição na tela.

O segundo profissional que pode ser envolvido neste processo é o User Experience (UX), que é um conhecedor dos costumes dos usuários da internet. Um bom UX sabe exatamente e como as pessoas procuram e consomem conteúdo, além do que procuram em um site, ou seja, ele saberá exatamente onde encaixar cada palavra, imagem e como destacá-las para maximizar a eficiência do site.

Após o trabalho destes dois profissionais, o desenvolvedor Front-End poderá entrar em ação. Ele irá receber todo este compilado de informação documentada e iniciará seu trabalho. Seu trabalho envolverá desenvolver tecnicamente de fato toda esta elaboração conceitual. Portanto, o Front-End irá escrever o código que fará o site nascer de fato. Em geral seu trabalho consiste em utilizar o HTML, CSS e Javascript para atingir o objetivo final idealizado no início do projeto.

Apesar de não ser o nosso foco, vale lembrar que existe também a posição do profissional de Back-End, que seria o outro lado da moeda do Front-End. Ele também se encarrega de codificar, porém o resultado de seu trabalho não é uma interface que podemos ver e navegar. Seu trabalho engloba montar os bancos de dados, desenvolver as lógicas e regras de negócio, garantindo que a informação esteja disponível para o profissional de Front-End colocar no site. Resumindo, quando você acessa um site, a estrutura que você vê é responsabilidade do Front-End, porém a origem, segurança e integridade da informação, é trabalho do Back-End.

1.2 História e evolução das linguagens

A evolução de uma página *web* ou site tem passado por diversas revoluções ao longo dos anos. No início uma página era desenvolvida apenas utilizando o HTML, os textos e imagens eram colocados em sua estrutura e pronto, tudo se resumia a isso. Um conteúdo estático que apresentava informações sobre empresas, notícias ou páginas pessoais.

Não demorou muito para que diversas novas tecnologia surgissem, porém as que perduraram e continuam se atualizando constantemente são o próprio HTML e seus companheiros **CSS** e **Javascript (JS)**. Segue abaixo um quadro com suas versões e datas de atualização:

	Surgimento	Última versão
HTML	1993 (HTML, HTML2, HTML3, HTML4, XHTML)	HTML5
CSS	1996	CSS4
Javascript	1995	2018

1.3 Como um navegador funciona

O navegador, ou *browser*, é o responsável pela exibição de todo conteúdo desenvolvido por um profissional de Front-End, por isso é essencial que se saiba como eles funcionam. Um navegador é basicamente um interpretador de código HTML, CSS e Javascript. Ele consegue diferenciar internamente o que é um texto padrão a ser exibido na tela e o que é um código que deverá ser interpretado antes de sua execução. Existem diversos navegadores desenvolvidos por empresas diferentes, porém existe uma norma que deve ser seguida por todas para garantir que as páginas sejam exibidas de forma similar em cada um deles. O órgão W3C é o grande responsável por manter este padrão da *web* e garantir que algumas normas sejam respeitadas, a fim de exibir sempre o melhor conteúdo de forma segura aos usuários.

Em resumo, pode-se dizer que um navegador não passa de um mero interpretador de códigos. Os que fazem isso de forma mais rápida e fiel, são os que se destacaram no mercado.

CAPÍTULO

02

HTML

2.1 Estrutura de uma página HTML

O primeiro e mais básico requisito para um documento ser HTML é ser salvo com a extensão ".html", afinal é o primeiro passo para que o navegador possa interpretá-lo, o segundo requisito é sua estrutura básica de código. Estes códigos são conhecidos como *tags* e são textos específicos utilizados durante o desenvolvimento, que ao serem carregados por um navegador, são interpretados e executados em alguns procedimentos, não apenas impressos na tela. Veja abaixo a estrutura básica:

```
<html>
        <head></head>
        <body></body>
<html>
```

Um código HTML pode ser escrito em qualquer *software* de edição de texto, como por exemplo o Bloco de Notas, desde que sempre seja salvo com a extensão ".html".

Mesmo com tanta liberdade, sempre recomenda-se utilizar um editor de texto apropriado para este tipo de trabalho. Atualmente os mais populares tem sido o **Visual Studio Code** da Microsoft, o **Sublime** da Sublime HQ ou até mesmo o **NotePad++**, que é *open source*. Todos estes são gratuitos e livres para uso.

2.2 Primeiras tags

As *tags* são palavras-chave específicas que fazem com que o navegador compreenda que alguma coisa deverá ser interpretada e montada na tela, após ler o arquivo HTML. Veja os exemplos abaixo:

	Cria um título, com numeração que pode variar de 1 a 5. 1 para título grande e 5 para pequeno
	Cria um parágrafo
 	Cria uma quebra de linha

Exemplo:

```
<h1>Meu título</h1>
<h3>Meu subtítulo título</h3>
<p>Meu texto</p>
<br>
<p>Meu outro parágrafo</p>
```

Meu título

Meu subtitulotítulo

Meu texto

Meu outro paragrafo

2.3 *Tags*, atributos e elementos

DIFERENÇA ENTRE *TAG* E ELEMENTO

A tag é o código escrito, literalmente, como por exemplo: *"<h1>"* ou *"<p>"*. Já o elemento faz referência a um trecho maior, envolvendo a tag escrita e o conteúdo textual que a acompanha, veja abaixo:

```
<p>Meu texto</p>
```

O elemento é toda a estrutura, inclusive podendo conter outras tags, construindo um conteúdo ainda maior, como veremos mais adiante. O ponto principal é que o elemento se refere do ponto de abertura de uma *tag* até o seu ponto de fechamento, incluindo tudo que está dentro deste trecho.

Como funcionam os atributos

Atributos são complementos de uma *tag* que possibilitam dar mais características a ela. Geralmente são adicionados dentro do espaço da *tag* no momento da sua abertura.

```
<p style="color:red">Meu texto.</p>
```

Dentro da *tag "<p>"*, adicionamos o atributo *"style='color:red'"*. Este atributo deixará o texto na cor vermelha. Claro que o atributo deve estar de acordo com o conteúdo da *tag*, afinal não faz sentido, por exemplo, colocar este atributo em uma *tag* que não contenha um texto. Por isso, os atributos podem variar muito de acordo com a *tag* que acompanham. Em alguns casos o uso do atributo pode ser obrigatório, como por exemplo em *tags* de imagens, que veremos mais para frente.

2.4 Tag textual

Formatação de textos

Existem diversas tags que são utilizadas para formatação de texto. Nos referimos a elas como tags textuais e as utilizamos com intuito de colocar uma formatação específica em um trecho de um texto, ou até em um conteúdo completo. Coloque as tags abaixo em um documento HTML para ver o resultado de cada uma delas:

Meu texto Meu texto	Texto negrito. As duas *tags* têm a mesma função
Meu texto	Texto itálico
Meu texto	Texto com *emphasis*
<mark>Meu texto</mark>	Texto marcado
Meu texto	Texto pequeno
~~Meu texto~~	Texto deletado
<ins>Meu texto</ins>	Texto inserido
_{Meu texto}	Texto alinhado na base
^{Meu texto}	Texto alinhado no alto

Quebra de linhas

Conforme vimos no primeiro exemplo de *tag* que mostramos, a *tag* responsável por quebrar linha é a
. Esta *tag* pode ser repetida inúmeras vezes, dependendo do tamanho do espaço que desejamos entre ambos textos.

Uma outra forma de realizar a quebra de linha é através da *tag <pre>*, que nos possibilita utilizar o Enter

do teclado para indicar onde o texto deve ser quebrado. Com esta *tag* não precisamos colocar um *
* no final das linhas.

 	Quebra de linha
<pre>Meu texto longo Loren ipsun salen lae Ipsun ratif loren salm. </pre>	Quebra de linha utilizando *<pre>*. O texto respeitará as quebras indicadas

Títulos e subtítulos

As *tags* de títulos e subtítulos são muito utilizadas, pois possuem uma pré formatação aplicada pelo próprio navegador, que aumenta ou diminui o tamanho do texto, de acordo com a opção utilizada.

O tamanho específico de cada título pode ser definido com o uso de CSS. Porém, mesmo se você pretende alterar completamente o tamanho, é importante ainda sempre utilizar estas *tags* para se referir a um título ou subtítulo.

Titulo	Título grande e de maior importância
Titulo	
Titulo	
Título	
Título	
Titulo	Título pequeno e de menor importância

2.5 Tag de conteúdo

Criando blocos de conteúdo

Utilizamos as *tags* de blocos para agrupar um conteúdo que faz referência a uma mesma área, podendo este conteúdo pode ter outras *tags* e textos envolvidos. Muitas vezes esta técnica é apenas para a organização do código e acaba não causando nenhum efeito visual específico.

Algum texto	Utilizado para agrupar trechos que contêm outras tags
Algum texto	Utilizado para agrupar textos

As *tags* acima não aplicam nenhuma alteração quando utilizadas sem os seus atributos, podendo elas ter os atributos *style*, *class* ou *id*. *Style* possibilitará aplicarmos alguma formatação de estilo à *tag*. Já *class* e *id* são atributos que podem ser utilizados em conjunto com CSS e Javascript. Por exemplo, o trecho abaixo criará um quadrado com fundo verde, texto branco e um espaço entre a borda e o texto de 20px:

```
<div style="background-color:green;color:white;padding:20px;">
        Loren ipsun salin loren ipsu more.
</div>
```

Loren ipsun salin loren ipsu more.

No caso do atributo *span*, a alteração é aplicada geralmente a trechos de texto:

```
Proin at ullamcorper <span style="color:red">metus interdum</span> et malesuada.
```

Proin at ullamcorper metus interdum et malesuada.

Parágrafos

A tag de parágrafo é muito utilizada em quase todo texto livre. Uma pré formatação será adicionada a ela pelo próprio navegador, além de um espaçamento inicial. Estas formatações iniciais podem ser alteradas utilizando CSS. É importante destacar que cada navegador poderá exibir com uma pequena diferença tais formatações.

Parágrafos não aplicam quebras de linha automáticas no meio do seu texto, logo devemos colocar alguns “
”, ou utilizar a opção <pre> para realizar a quebra de linhas, como já vimos anteriormente.

```
<p>Meu parágrafo</p>
<p>Meu segundo parágrafo</p>
<p>
Meu parágrafo
grande sem quebras
De linha.
</p>
```

Meu paragrafo

Meu segundo paragrafo

Meu paragrafo grande sem quebras de linha.

2.6 Links e imagens

Um link é utilizado para criar um ponto onde o clique do mouse levará para uma outra página. Podemos definir links em textos ou imagens, para isto utilizamos a tag “<a>”.

Meu link	Cria um link em um texto

Esta tag tem o uso obrigatório de um atributo “href=”. Com ele definimos o local para onde o usuário será redirecionado após clicar no link, que pode ser dentro do próprio site ou algum outro endereço externo.

```
<a href=http://www.google.com.br>Meu link para google</a>
```

O link acima redirecionará o usuário para a página do Google. Ao passar o mouse sobre o texto do link, o cursor se tornará um ícone de mão.

Links internos: quando queremos direcionar o usuário para outra página do nosso site não é necessário utilizar o endereço completo. Podemos colocar apenas o nome do arquivo da página: Contato

Links externos: para redirecionar o usuário para uma página externa ao nosso site, devemos proceder como no exemplo do link do Google, já demonstrando que o endereço completo deve ser especificado.

Atributo target: um outro atributo que não é obrigatório, mas é muito importante, é o target. Ele define se a nova página direcionada pelo link será aberta na página atual ou em uma nova aba.

 para google	Abre o link em uma nova página
 para google	Abre o link na mesma página

2.7 Áudios e vídeos

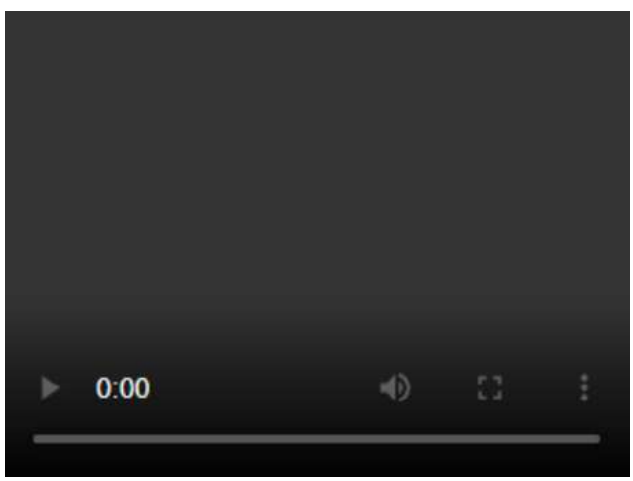
Com o HTML5, ganhamos também a opção de adicionar conteúdo mais dinâmico em um site. Em versões anteriores do HTML, para adicionar um vídeo ou áudio a uma página, era necessário utilizar um plug-in de terceiros ou um código Javascript. Na versão 5, temos tags específicas para este tipo de conteúdo.

Para adicionarmos um áudio a um site, temos que seguir a estrutura abaixo:

```
<audio controls>
        <source src="audio.mp3" type="audio/mpeg">
        Não foi possível carregar o áudio.
</audio>
```

Para vídeos, utilizamos esta:

```
<video width="320" height="240" controls>
        <source src="movie.mp4" type="video/mp4">
        Não foi possível carregar o vídeo.
</video>
```



A propriedade controls é responsável por dizer ao navegador que um controle de mídia deverá ser montado na tela. No caso do vídeo, as propriedades de width e height devem ser especificadas para a montagem da área de exibição do vídeo com o tamanho desejado. Source é a propriedade que define o caminho do arquivo a ser executado.

Para as estruturas de conteúdo, áudio e vídeo, é necessário colocar o seu arquivo fonte dentro de uma pasta no seu projeto e referenciar o caminho até ele. O próprio navegador fica responsável por montar a interface para execução.

Também é possível referenciar mais de um arquivo para reprodução, para isto basta duplicar a linha source. Mesmo que um dos arquivos acrescentados não carregue, isso não afetará os outros arquivos, pois as linhas são independentes. Já se nenhum dos arquivos carregar, a mensagem descrita no final será exibida. É necessário ter bastante cuidado, pois é muito comum navegadores terem problemas para exibição de mídias.

2.8 Tabelas e listas

Tabelas

As tabelas são um grande recurso do HTML e um dos mais utilizados desde sua primeira versão. Além de possibilitar que exibam algum conteúdo em forma de tabela, também podemos utilizá-la para apoiar no layout. Seu uso é vasto e suas tags combinadas com CSS, podem criar estruturas bem avançadas.

A estrutura básica de uma tabela é composta por três tags, veja abaixo:

<table></table>	Define onde inicia e termina a tabela
<tr></tr>	Cria uma linha
<td></td>	Cria uma coluna
<th></th>	Cria linha do cabeçalho (opcional)

Para estruturar uma tabela podemos utilizar as seguintes linhas de código:

```
<table>
    <tr>
        <th>Dia</th>
        <th>Valor</th>
    </tr>
    <tr>
        <td>01/01/2019</td>
        <td>R$10,00</td>
    </tr>
    <tr>
        <td>05/01/2019</td>
        <td>R$25,00</td>
    </tr>
</table>
```

Dia	Valor
01/01/2019	R$10,00
05/01/2019	R$25,00

Podemos observar que o resultado será uma tabela com duas linhas e duas colunas de conteúdo, além de uma linha de cabeçalho.

Outras tags ou atributos podem ser utilizados para compor esta estrutura. Assim como na tag div, a tabela não exibirá bordas caso isto não seja especificado no código. Sua formatação deve ser definida dentro da tag "<table>", para assim ser aplicada em toda estrutura.

```
<table border=2px cellspacing=1px cellpadding=1pxwidth=2pxalign=center bgcolor=#fefefe bordercolor = black>
```

Dia	Valor
01/01/2019	R$10,00
05/01/2019	R$25,00

Tags importantes para desenhar uma tabela:

Border: Define a espessura das bordas da tabela;
Cellspacing: Define o espaçamento entre cada célula;
Cellpadding: Define o espaçamento entre o conteúdo da célula e sua borda;
Width: Define a largura da tabela;
Align: Define o alinhamento da tabela em relação à página;
Bgcolor: Define a cor de fundo da tabela;
Bordercolor: Define a cor das bordas da tabela.

É possível também aplicar uma formatação especificamente para uma linha ou coluna da tabela, para isto basta adicionar estes elementos nas tags: "<tr>" e "<td>". Lembramos que algumas propriedades são apenas válidas para estas tags, como por exemplo as abaixo:

```
<td bgcolor=#efefef
bordercolor=#cccccc nowrap
colspan=2 rowspan=2>
```

Nowrap: Define a não quebra de linha caso o texto seja longo demais, nas células da tabela;
Colspan: Especifica o número de colunas a serem mergeadas;
Rowspan: Especifica o número de linhas a serem mergeadas.

Quando trabalhamos com tabelas muito complexas, podemos utilizar o apoio de ***agrupadores***, que facilitam o trabalho de definição de layout. Para isso, podemos utilizar as tags ***thead***, ***tbody*** e ***tfoot***. Elas respectivamente representam qual é a região do cabeçalho, corpo e rodapé da tabela. Tais tags não afetarão em nada o layout de sua tabela, caso não seja adicionado um CSS para elas.

```
<table>
        <thead>
                <tr>
                        <th>Month</th>
                        <th>Savings</th>
                </tr>
        </thead>
        <tbody>
                <tr>
                        <td>January</td>
                        <td>$100</td>
                </tr>
                <tr>
                        <td>February</td>
                        <td>$80</td>
                </tr>
        </tbody>
        <tfoot>
                <tr>
                        <td>Sum</td>
                        <td>$180</td>
                </tr>
        </tfoot>
</table>
```

Listas

As listas são outra forma de estruturar conteúdo dentro de uma página. Seu uso é bastante versátil, pois podem ser utilizadas para listar itens, imagens ou até mesmo para montagem de menus e exibição de notícias, entre outras funções. Existem três tipos de listas: ordenadas, não ordenadas e listas de definição.

Ordenadas: são listas onde conseguimos definir qual é a sequência dos números que queremos organizados. Elas utilizam uma tag principal para a abertura da estrutura e uma tag interna para a definição do conteúdo. Também podemos utilizar alguns outros atributos, veja abaixo:

	Cria a estrutura da lista
	Representa um item na lista
Type	Define o tipo de marcador. 1, A, a, I ou i
Start	Define o número de início da contagem
Reversed	Define uma contagem regressiva

```
<ol type="1" start="10" reversed>
    <li>Item</li>
    <li>Item</li>
    <li>Item</li>
</ol>
```

10. Item
9. Item
8. Item

Não ordenadas: são listas que possuem uma estrutura similar às ordenadas, porém os marcadores dos itens não são contadores, portanto não é possível ordenar a lista. Pelo mesmo motivo, as propriedades "Start" e "Reversed" não são aplicáveis. Os marcadores podem ser definidos como

“square”, “circle” ou “disc”, veja um exemplo abaixo:

```
<ul type=”disc”>
    <li>Item 1</li>
    <li>Item 2</li>
</ul>
```

- Item 1
- Item 2
- Item 3

Listas de Definição: Este tipo de lista é mais utilizado quando queremos montar algo similar a um glossário. Nela adicionamos tópicos e a sua representação no navegador será como uma lista, porém sem marcador algum.

```
<dl>
    <dt>Google</dt>
    <dd>Uol</dd>
    <dt>Wikipedia</dt>
    <dd>Catraca Livre</dd>
</dl>
```

Google
    Uol
Wikipedia
    Catraca Livre

2.9 Tag semântica

A semântica é o estudo do significado das palavras, desta forma, a internet tem focado em criar páginas com tags que dizem por si só qual o tipo de conteúdo que contém. Assim os navegadores e motores de busca conseguem encontrar e ler as páginas de forma mais fácil e rápida.

Páginas com tags semânticas se destacam das demais e podem ser mais bem ranqueadas nos resultados de busca. É muito provável que um analista SEO recomende utilizar este tipo de tag em um site.

Não semântico	Semântico
 	<article> <aside> <details> <figcaption> <figure> <footer> <header> <main> <mark> <nav> <section> <summary> <time>

As tags “div” e “span” não dizem nada sobre o seu conteúdo, logo não são recomendadas para uso sozinhas ou como agrupadores maiores de um conteúdo. Por mais que o seu uso seja constante e muito útil, deve-se sempre considerar utilizar este tipo de tags dentro de uma estrutura com tags semânticas.

Veja como exemplo um código para a criação de um artigo em uma página da web:

```
<article>
    <header>
        <h1>Título do meu artigo</h1>
<p>Subtítulo ou descrição</p>
    </header>
    <p>
            Conteúdo do meu artigo.
.   </p>
</article>
```

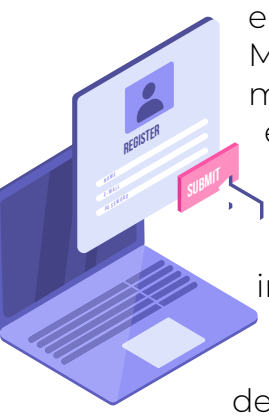
Agora um menu ou uma lista de links:

```
<nav>
    <a href="/html/">HTML</a> |
        <a href="/css/">CSS</a> |
        <a href="/js/">JavaScript</a> |
        <a href="/jquery/">jQuery</a>
</nav>
```

Terminamos com uma figura:

```
<figure>
            <img src="minhaFoto.jpg"
            alt="Texto no mouse over">
            <figcaption>Minha
            legenda<figcaption>
</figure>
```

2.1.1 Formulário



O HTML também é utilizado na construção de sistemas internos e externos para empresas. Muitos destes sistemas utilizam menos requisitos de layout e mais de funcionalidades, para esta situação, as tags para formulários são muito úteis. No caso de sites para internet, geralmente as tags de formulários são mais utilizadas precisamente para formulários de contato.

Tudo que se refere a um formulário deve sempre estar dentro da tag "<form>", que é responsável por mostrar ao navegador onde está o conteúdo referente ao formulário. Para que um formulário funcione adequadamente, é necessário o uso de uma linguagem de Back-End, que executará alguma função programada. Por enquanto, vamos apresentar apenas os itens necessários para a execução de um formulário na tela.

Campos para inserção de texto

Existem diversas opções de campos, podendo variar em propriedades e formas de escrita. Abaixo mostramos um campo de inserção de texto e suas opções de variação:

<input type="button">	<input type="file">	<input type="radio">
<input type="checkbox">	<input type="hidden">	<input type="range">
<input type="color">	<input type="image">	<input type="reset">
<input type="date">	<input type="month">	<input type="search">
<input type="datetime-local">	<input type="number">	<input type="submit">
<input type="email">	<input type="password">	<input type="tel">
<input type="text">	<input type="time">	<input type="url">
<input type="week">		

A tag "<input>" é a mais importante e principal responsável pela maioria dos campos, já o atributo type é o responsável por definir o tipo do campo. Para o uso de CSS e Javascript é também recomendado adicionar uma tag id e name em input. Para o caso de optar por deixar o campo preenchido, pode-se usar o atributo value.

Lista de propriedades

id="name"	Id do campo para uso no CSS ou JS
value="João Maria"	Valor que aparecerá dentro do campo como default
name="name"	Nome do campo para uso no CSS ou JS
readonly	Campo apenas para leitura
disabled	Campo desabilitado na tela
size="40"	Especifica o tamanho da caixa do campo na tela
maxlenght="10"	Máximo de caracteres no preenchimento do campo
autofocos	Deixa o campo selecionado ao carregar a tela

Veja mais um exemplo:

```
<input type="text" name="nome" id="name" value="João Maria">
```

Botões

Outra função importante da tag input é a inserção de botões, com as propriedades de id, class e name também podendo ser preenchidas. Também é possível adicionar um botão através da tag "<button>", que possui os mesmos atributos, porém exige que para aparecer o texto no corpo do botão, se inclua a palavra entre a tag de abertura e a tag de fechamento.

```
<button type="button" id="meuBotao" class="minhaClasse" >Salvar</button>
```

Post e Get

Normalmente, após o preenchimento de um formulário, é necessário clicar no botão para realizar o envio desta informação para algum lugar. As palavras post e get se referem justamente a forma como os dados são enviados para o Back-End. Este processo é algo simples para o usuário final do site, porém interfere diretamente na forma que os desenvolvedores vão trabalhar.

Get: os dados são enviados via URL para o Back-End ou para outra página (receptora em geral). Se acessarmos esta URL que estamos enviando, veremos algo similar ao mostrado abaixo:

```
...paginadestino.html?nome=JoaoMaria&sobrenome=Silva&idade=35
```

Se repararmos, os dados de todos os inputs da tela são colocados no texto da URL, junto de seus ids. Não é recomendado o uso deste método quando se trafega muitos dados, pois a URL possui um limite de 255 caracteres.

Post: Os dados são enviados de forma transparente, como se estivessem dentro de um envelope no navegador. Este método pode ser utilizado quando são muitos os campos a serem preenchidos no formulário. Precisamos ressaltar que não ter os dados na no corpo da URL não significa que estes estejam mais seguros, são apenas formas diferente de envio. Os dados do envelope também podem ser acessados via navegador através do inspecionador de elementos.

O uso de cada uma dessas opções é determinado pelo seu objetivo em si. A tag get costuma ser utilizada quando queremos buscar alguma informação no banco de dados e para isso precisamos enviar algum id ou chave. Já quando precisamos salvar alguma informação, então utilizamos a tag post. Apesar de se poder executar ambas as funções com o mesmo método, a recomendação é que se use o adequado para o objetivo sempre.

Por último, para definir a forma de envio do dado de seu formulário, existe a tag <form>. Após o clique no botão de ação, a form respeitará a propriedade para fazer o envio.

```
<form method="post"> ou <form method="get">
```

CAPÍTULO

03

USER EXPERIENCE (UX)

3.1 a. O que é UX Design?

UX é uma sigla para “User Experience” em inglês, ou “experiência do usuário” como é utilizado em português. É um termo criado por Donald Norman nos anos 90, quando ele era vice-presidente na Apple Advanced Technology Group. O UX foi se popularizando, até nos dias de hoje se tornar uma disciplina que ajuda profissionais (principalmente designers) a desenvolverem produtos ou soluções que resolvam problemas e gerem valor para as pessoas.

Como a própria palavra “experiência” sugere, trata-se de todo o contato que alguém tem com um produto: como se sente em relação a ele, como ele funciona, como se comporta durante a interação, se existe alguma dificuldade ou se é fácil de usar, como ele entrega a solução que se propõe a fazer e como o usuário se sente após atingir um objetivo ou resolver alguma necessidade.

“O design está em toda parte. Você não usa nada que não tenha sido projetado de alguma forma. O design te ajuda a executar uma tarefa, é só uma questão se o design é bom ou não. Não é só o visual, é toda a experiência.” Ian Spalter, Head de Design no Instagram

O produto não precisa nem ser necessariamente digital, como um site ou aplicativo, a experiência do usuário pode ser empregada em objetos físicos (um garfo, por exemplo) ou serviços em geral (como a contratação de banda larga para sua casa).

No nosso caso, o foco será em produtos digitais e em como conceitos de UX poderão te ajudar a desenvolver produtos melhores e mais intuitivos, considerando sempre o usuário e suas necessidades no centro da experiência.

Já se foram os tempos em que desenvolvedores se preocupavam apenas em executar o que designers (ou web designers) projetavam. Se você é um desenvolvedor com conhecimentos (mesmo que básicos) de experiência do usuário, você já está à frente, assim como os designers estão se preocupando mais em aprender a codificar, para projetarem coisas viáveis, minimizando chances de limitações técnicas. Tudo isso torna o trabalho muito mais sincronizado e qualificado, refletindo positivamente nos usuários e nos negócios.

Se formos colocar em etapas como acontece um processo de UX, ele seria dividido em: **Descobertas, Design, Teste, Lançamento** e **Observação pós lançamento.**

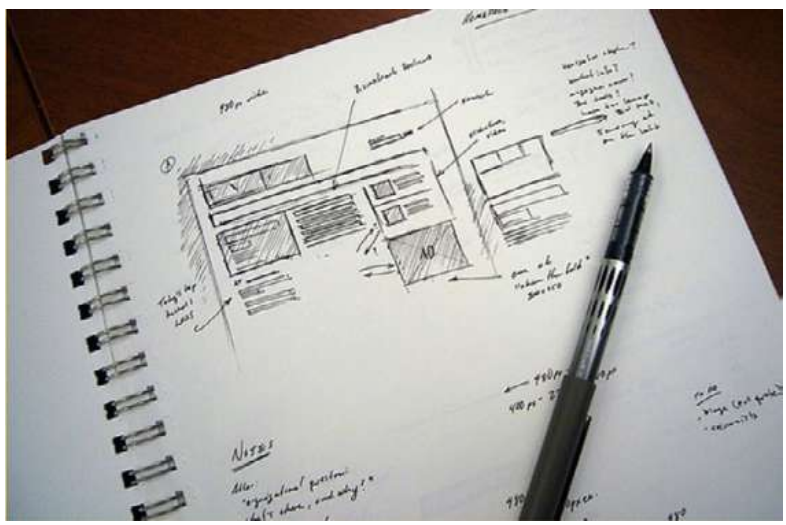
Descobertas: é o momento em que o designer investiga, faz pesquisas, benchmarking com outras empresas similares, conversa com usuários e pessoas da empresa em relação ao produto ou serviço, observa comportamentos e identifica dores dos usuários, coleta insights até finalmente descobrir o real problema que precisa ser solucionado.

Design: após descobrir o problema, é hora de colocar a criatividade e o know how em prática. É o momento de pensar como aquele problema será resolvido. Pode ser uma nova jornada de cadastro, alguma melhoria na jornada existente, um fluxo mais fluido, uma interface mais acessível, diminuir etapas e manter apenas informações essenciais, melhorar algum ponto que talvez esteja fazendo os usuários abandonarem seu produto etc. Uma infinidade de soluções pode surgir e para economizar

esforços de desenvolvimento e design, otimizando a validação, é comum que a primeira versão da solução seja projetada em baixa fidelidade, como um rascunho, wireframe ou, dependendo do projeto, um protótipo, que é uma opção mais fiel ao produto que será projetado posteriormente.

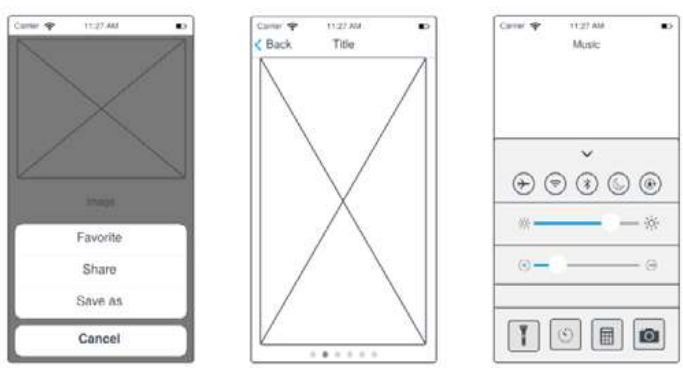
Veja abaixo alguns exemplos de entregáveis:

Rascunho: também conhecidos como "rabisco frame", os rascunhos são rabiscos estilizados em papel, resumindo as partes mais importantes da interface e da interação entre telas.



Fonte: brasil.uxdesign.cc

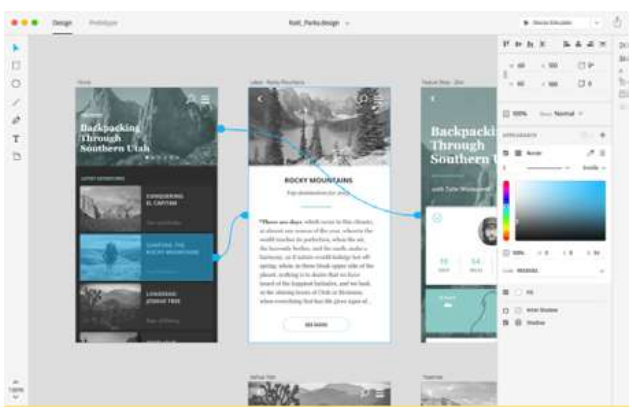
Wireframe: é mais uma versão de rascunho, porém digital e mais estruturada. No wireframe, é possível visualizar mais fielmente toda a disposição de informações que estarão na interface, como botões, títulos, caixas de texto, imagens, campos de digitação e quantidade de telas.



Wireframe de aplicativo construído no LucidChart.
Fonte: https://www.lucidchart.com/pages/templates/wireframe/mobile-wireframe-template

Protótipo:

O protótipo é a versão mais fiel da interface do site ou app, criado através de um software gráfico, oferecendo a possibilidade de interação real entre as funcionalidades do site ou app, ainda na fase de testes. Desta maneira, é possível criar um design de site 100% fiel ao que irá ao ar, com todo o fluxo de interação inserido nele. Geralmente este formato pode ser entregue para um usuário interagir, acessar os botões, ser direcionado para as outras telas, completar uma tarefa, realizando assim um teste muito mais amplo do site, antes de colocá-lo no ar.



Fonte: http://techcrunch.com/2016/03/14/adobe-launches-experience-design-cc-a-new-tool-for-ux-designers

Tanto o wireframe, quanto os protótipos podem ser projetados em softwares como: Adobe Xd. Sketch, Photoshop, UX Pin, InVision, Zeplin, Figma, Visio, Marvel, Axure RP, Wireframe.cc, LucidChart, Miro e muitos outros. Escolher entre estes softwares inclui pensar vários fatores, já que além de alguns serem gratuitos e outros não, diferenças aparecem na possibilidade de utilização online diretamente através do browser, funcionalidades de prototipação (como vincular interação de telas e simular navegação) e até compatibilidade com os diferentes sistemas operacionais (Windows, iOS ou Mac). Provavelmente, em algum momento você irá receber um projeto feito em algum desses softwares para desenvolver.

Teste

Chamado de "teste de usabilidade", é o momento em que o protótipo é entregue ao usuário para teste e observação. É importante ressaltar que o usuário poderá ter ações imprevisíveis no momento que interagir com a interface, então não se frustre caso o usuário clique em um botão que não deveria, é importante anteceder estes possíveis cenários para fazer ajustes antes de lançar no mercado. Quanto mais cenários inesperados forem catalogados e melhorados durante o teste, mais perto de um possível "site perfeito" estará a entrega final.

Desenvolvimento e lançamento

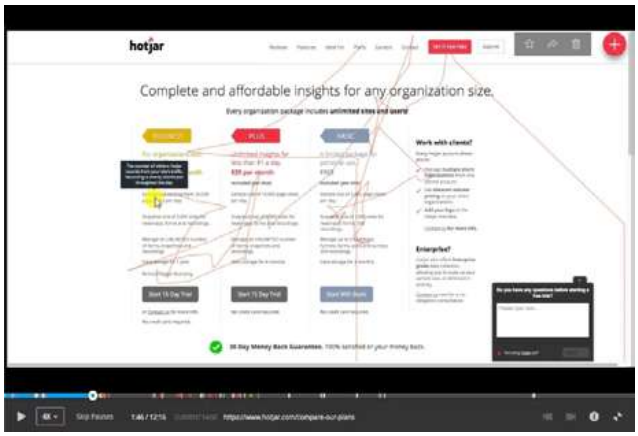
Feitos todos os testes de usuário, anotações e ajustes no layout do protótipo, é a hora dos desenvolvedores entrarem em cena e criarem as telas para levar o serviço ao ar.

Observação pós lançamento

Mesmo após os testes de usuário, para prever cenários e melhorar a interface antes de partir para o código, é importante continuar observando o comportamento e avaliando a performance após o lançamento. Afinal, uma gama maior de usuários usará o serviço e novos desafios de interface, experiência e desenvolvimento podem surgir. Um site ou app nunca será 100% perfeito. O comportamento humano é tão imprevisível que sempre haverá oportunidades de melhoria.

Existem alguns serviços que podem ajudar a identificar melhorias na interface, chamados de serviços de observação online. Nestas observações são feitas leituras de cursor do mouse para entender como o usuário se orien-

ta, mapas de calor para saber onde ele olha ou permanece mais tempo na tela e diversas outras funcionalidades que podem ajudar na tomada de decisões para melhorar o serviço. Algumas das ferramentas são: Hotjar, MouseFlow ou Mopinion.



Aqui é possível mapear os movimentos do cursor do mouse do usuário. Fonte: www.hotjar.com/tour

3.2 b. O que é UI Design?

UI é a sigla para "User Interface" (Design de Interface do Usuário), que é a forma interativa e visual que uma pessoa interage e controla algum dispositivo, site, aplicativo, software etc. Geralmente o controle é feito através de recursos visuais, como menus, botões, campos de preenchimento, seleção de campos e outros. A orientação é exibida através de cores, textos, calls to actions (CTA), mensagens de feedback, transições entre ações e outros recursos que ajudam o usuário a percorrer a jornada de uso do serviço.

A seguir, veja um exemplo básico que ilustra uma pequena interação cotidiana de criação ou redefinição de senha. Observe que, com algumas simples "dicas" que a interface mostra ao usuário, ele consegue criar uma senha sem muitas barreiras. Vamos supor que um aplicativo de compras peça uma senha com:

- Pelo menos uma letra maiúscula;
- Pelo menos uma letra minúscula;
- Pelo menos um caractere especial;
- Pelo menos um número.

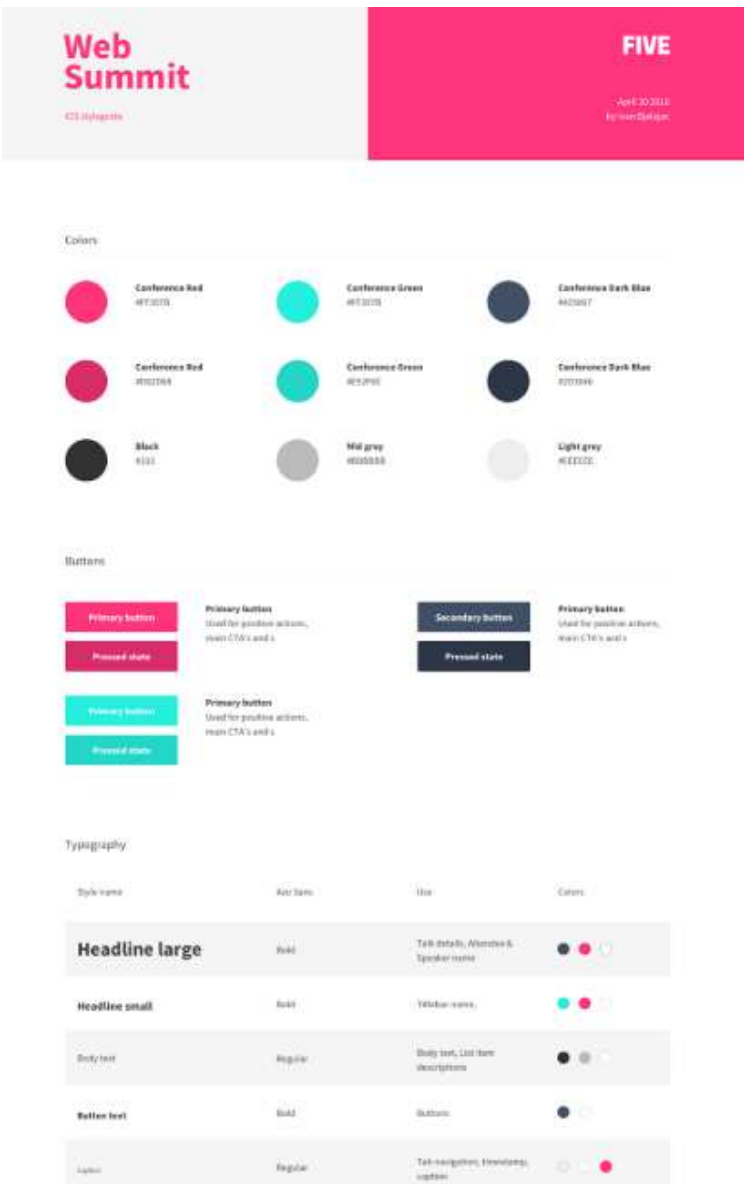
É importante que a interface antecipe possíveis cenários de erro ou ações que o usuário possa tomar

A experiência do usuário flui muito melhor quando a interface mostra as "dicas" (que podem ser necessárias por questões internas de segurança ou sistema) no momento da criação de senha. É possível que, pelo menos uma vez na sua vida, você já tenha tentado criar uma senha nova para algum site ou serviço, e só depois de apertar o "OK", a interface te dê feedbacks como: "sua senha precisa de, no mínimo, 6 caracteres", "senha muito curta", "você precisa de pelo menos um número" e etc. Se o usuário já sabe previamente os pré-requisitos de sua nova senha, não terá que pensar duas vezes em maneiras de respeitar os requisitos do sistema.

Deixe tudo claro o mais rápido possível, com um design que indique: onde o usuário está, o que acabou de acontecer, o que ele pode fazer, o que vai acontecer quando ele fizer alguma ação e por fim, quando a tarefa terminar, o usuário precisa estar seguro de que terminou uma etapa.

Style Guide

Style Guide é um guia visual contendo os elementos de interação da interface, como tipografia, paleta de cores, estilos de botões, caixas e formatação de texto, ícones, tamanho de fonte, tratamento de imagem e qualquer outro elemento que deva compor a interface. É como se fosse uma "bandeja de itens visuais padronizados" que devem ser respeitados no momento de criação e desenvolvimento de uma interface. A padronização é importante para manter uma linha visual consistente, independentemente da plataforma ou design responsivo a ser aplicado.



Modelo de Style Guide do Web Summit.
Fonte: dribbble.com/ibjelajac

Dentro de uma ordem de processos de design, um Style Guide entraria em:

Style Guide > wireframe > protótipo > código.

É importante que um projeto tenha uma identidade visual coerente e consistente, pois isso facilita para o designer adaptar o estilo para diversos tipos de plataformas responsivas, fazendo com que o desenvolvedor tenha também facilidade em desenvolver dentro do padrão de cores e formatos. O Style Guide serve também para o usuário final se orientar dentro de uma interface e conseguir compreender a hierarquia das informações sem fazer muito esforço cognitivo.

3.3 Qual é a diferença entre UI e UX Design?

É muito comum vermos as siglas "UI/UX" juntas, como se fossem a mesma coisa. A verdade é que dentro de UI não necessariamente existe UX, pois pode se tratar apenas de um projeto visual (por exemplo, um redesign). Quanto ao UX, é possível que dentro dele exista UI, caso a parte visual seja uma das soluções para melhorar a experiência do usuário. Enquanto um UX projeta a experiência e estuda o sentimento do usuário, o UI se encarrega de formatar as informações da melhor maneira possível, na interface do produto.

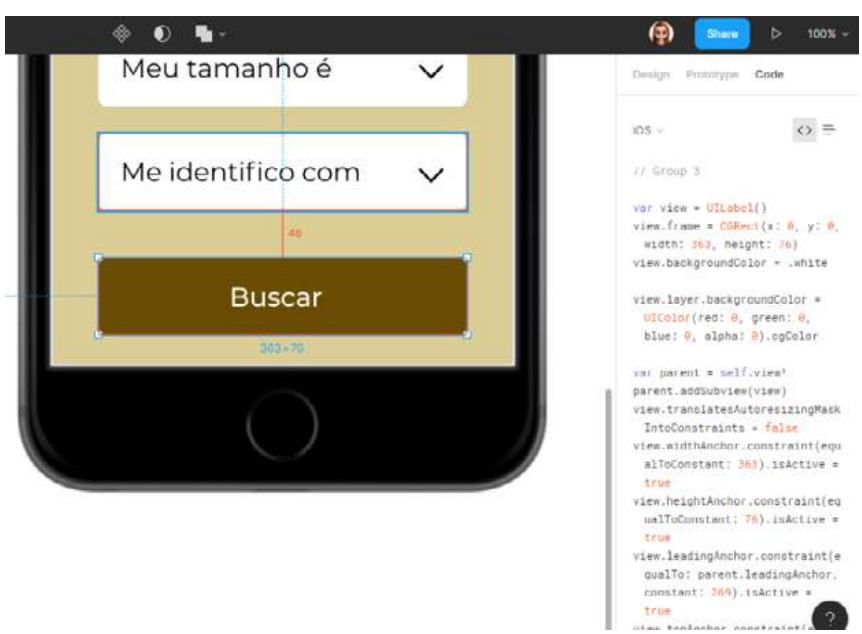
3.4 Como ferramentas de UX e UI Design podem ajudar em Front-End?

Quando uma interface é projetada em softwares como Figma, Zeplin, Sketch e outros, você pode selecionar os elementos separadamente, para ver propriedades como tamanho, cores hexadecimais e posicionamento. Isto facilita no momento de programar.

Abaixo, veja três exemplos de um layout construído no Figma. Observe que o botão “Buscar” está selecionado e é possível ver todas as propriedades do elemento em CSS, iOS e Android.

CSS

iOS

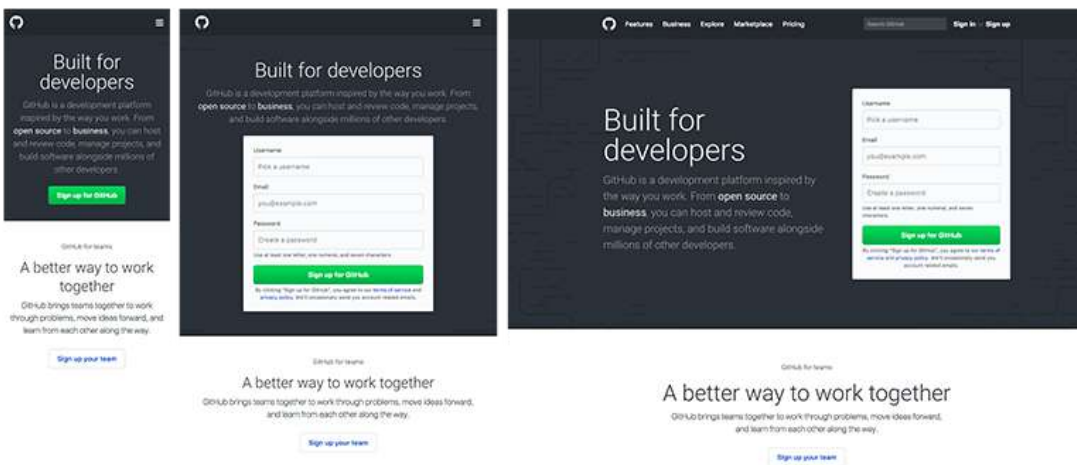


Android

3.5 Responsive Web Design

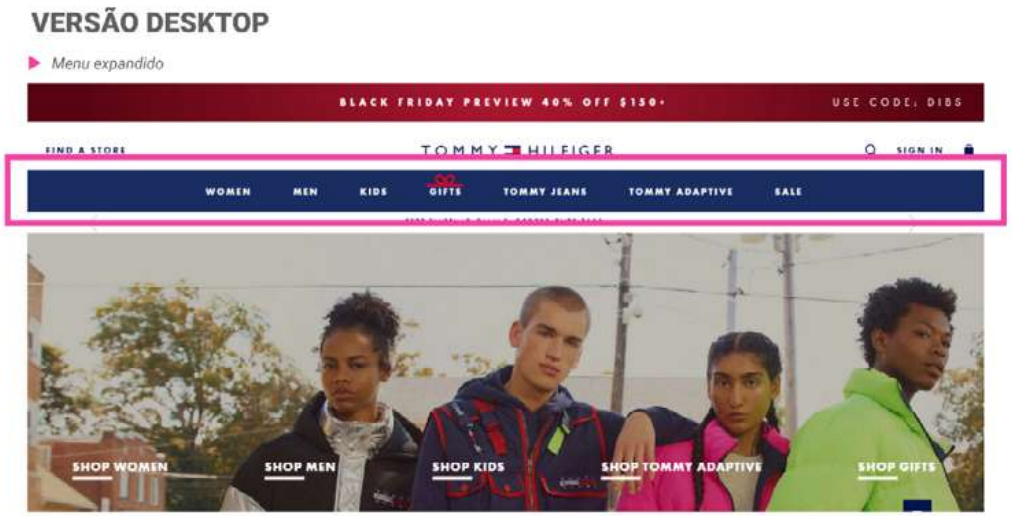
Quando falamos em design responsivo, significa que um site foi construído para ser acessível e fácil de navegar, independentemente do dispositivo que o usuário vai usar para acessá-lo, seja um smartphone, um notebook ou um iPad. Os elementos da interface são programados para se adaptarem ao tamanho das telas, respeitando também as condições de interação. Por exemplo, num smartphone os botões são maiores para serem mais acessíveis ao tamanho dos dedos, enquanto um botão num site de desktop é menor já que a "seta" do cursor é pequena. As imagens também são programadas para se ajustarem nas telas ou até removidas juntamente com outros elementos desnecessários que comprometam o carregamento de dados em dispositivos mobile.

Abaixo um exemplo de design responsivo do GitHub:

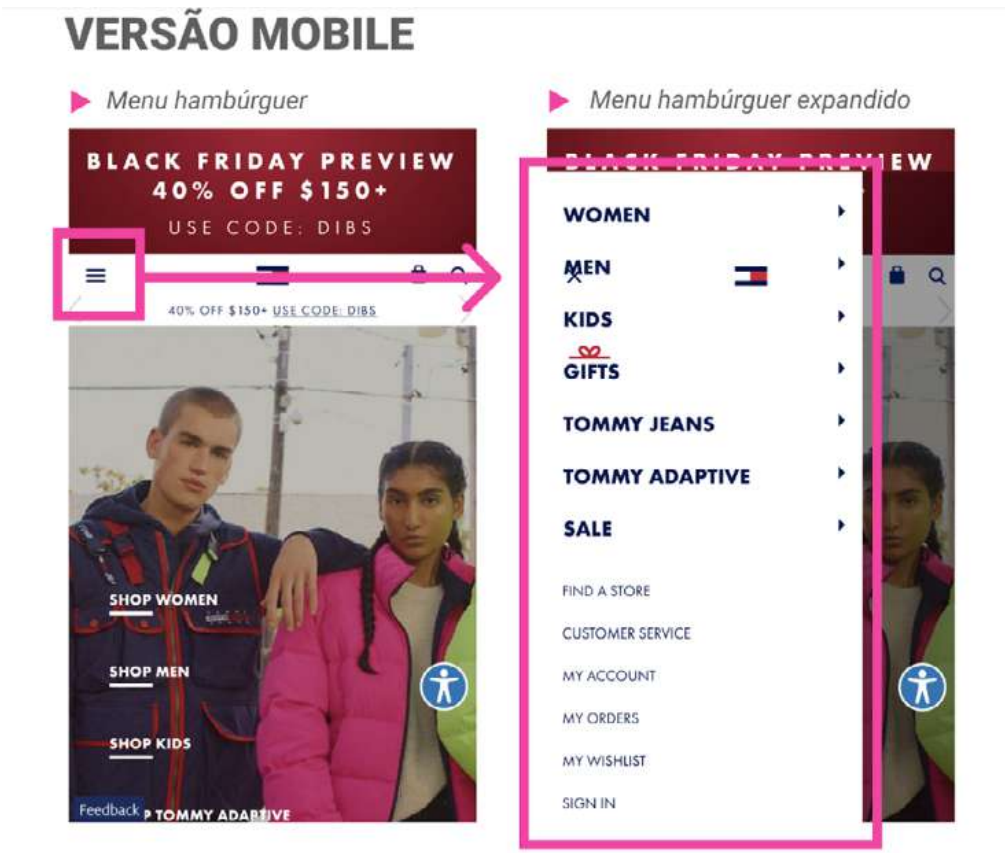


Fonte: www.invisionapp.com/inside-design/examples-responsive-web-design/

Podemos notar isso quando se acessa, por exemplo, um e-commerce em um desktop. Na home é possível ler todas as categorias de produtos do menu, área de login, carrinho de compras, barra de buscas expandida e vários banners. Descendo a página os produtos aparecem distribuídos em grade e muitos outros campos.



Porém ao acessar o mesmo site em um smartphone, as informações são distribuídas de forma mais resumida e verticalizada (uma vez que a navegação no mobile é mais restrita do que em um desktop, que tem uma área maior e horizontal). Outra questão é o contexto de uso do smartphone vs. desktop. Por exemplo, uma compra feita na fila do aeroporto pelo smartphone precisa ser mais otimizada e sem distrações, comparada a uma compra feita num desktop, onde possivelmente o usuário esteja sentado e com disponibilidade de tempo para navegar em mais telas e ser impactado por outras informações.



Com o aumento dos acessos a sites via smartphone, a prioridade em desenvolver uma interface com ótima performance mobile cresceu também. Isto deu início ao conceito de Mobile First, que nada mais é do que projetar um site atendendo primordialmente a navegação mobile, para depois pensar em desenvolver a versão desktop.

3.6 Design Thinking

Design Thinking é uma disciplina que foi popularizada por David M. Kelley, fundador da IDEO, uma das consultorias de design mais renomadas do mundo. Trata-se de um conjunto de metodologias utilizadas para solucionar problemas de negócios, baseando-se nas necessidades do usuário para criar novos produtos e serviços, visando inovar o projeto.

Um processo de Design Thinking é composto pelas seguintes etapas:

Criar empatia com o usuário: identificar as dores do usuário, o que querem, o que sentem, o que gostam e suas motivações;

Definir: depois de muita informação coletada, definir e entender qual é o problema que precisa ser solucionado;

Idealizar: é o momento de fazer brainstorming, colar post-its nas paredes e desenvolver ideias que sejam viáveis tecnologicamente, financeiramente e façam sentido para o usuário. Muitas empresas que facilitam processos de Design Thinking costumam reunir profissionais de várias áreas, como negócios, design, desenvolvimento, lideranças e o que mais fizer sentido para enriquecer a conversa e gerar uma colaboração entre disciplinas.

Prototipar: depois de muitas ideias e hipóteses geradas, é hora de escolher uma ou mais ideias que façam mais sentido e desenvolver algum protótipo que possa ser validado por usuários reais;

Testar: na hora de testar é importante observar e anotar todos os insights obtidos com o protótipo, que dificilmente será perfeito, mas se fizer sentido num primeiro momento ao usuário, você já estará no caminho certo.

Os processos de UX são bem parecidos com Design Thinking quando se trata de colocar o usuário no centro da experiência a fim de solucionar algum problema. É muito comum UX designers usarem Design Thinking para idealização, teste e prototipação antes de desenvolverem algo concreto.

Fonte: unsplash.com/@youxventures

3.7 Referências e materiais para se aprofundar mais em UX, UI e Design Thinking

Livros

- About Face: The Essentials of Interaction Design – *Alan Cooper, Robert Reimann, Christopher Noessel, David Cronin*
- Princípios do Web Design Maravilhoso – *Jason Beaird*
- Information Architecture for the World Wide Web – *Peter Morville*
- Não me faça pensar – *Steve Krug, Daniel Croce*
- Design Centrado no Usuário – *Travis Lowdermilk*
- Lean UX – *Jeff Gothelf, Josh Seiden*
- Change by Design – *Tim Brown*
- The Elements of User Experience – *Jesse James Garrett*

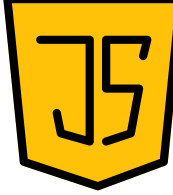
Blogs

- UX Collective: brasil.uxdesign.cc
- UI Lab: uilab.com.br
- IDEO Blog: ideo.com/blog
- Hackernoon: hackernoon.com
- Usability Geek: usabilitygeek.com
- Web Credible: webcredible.com/blog

CAPÍTULO

04

JAVASCRIPT



4.1 Introdução ao conceito

Javascrpt (JS) é a linguagem de Front-End mais utilizada na internet. Em conjunto com HTML e CSS ela é responsável por montar o comportamento de uma página.

- O HTML desenha a página;
- O CSS estiliza a página;
- O Javascript adiciona o dinamismo, e o comportamento

Apesar do nome, o Javascript não está relacionado com a linguagem de Back-End JAVA, trata-se apenas de uma coincidência de mercado. Seu lançamento foi em 1995 e logo se tornou padrão de mercado para desenvolvimento web. Isto se deve ao fato de ser de fácil entendimento e manipulação. O seu uso também não atrapalha o carregamento dos sites ou afeta a performance.

Esta é uma linguagem *case sensitive*, ou seja, a diferença entre letras maiúsculas e minúsculas afeta o seu comportamento. A extensão de um arquivo JS deve ser ".js" no momento de salvar, desta forma o navegador conseguirá reconhecê-lo. Ela também possui suporte para o uso de classes, funções e estruturas de condição e repetição, veremos isso no decorrer do texto.

Assim como o HTML, também pode-se utilizar qualquer editor de texto para manipulação de JS, porém se recomenda alguns softwares de mercado como o Visual Studio Code ou Sublime.

Muitos sites dão apoio ao uso desta linguagem, como por exemplo https://www.w3schools.com, que é o site oficial de JS, em relação a suporte ao conteúdo e versão.

Adicionando JS a uma página HTML

Códigos Javascript ficam em um arquivo com extensão .js que deve ser referenciado na página HTML. Desta forma eles poderão ser carregados e acionados quando necessário, para isso utilizamos a tag "<script>". Esta tag é recomendada ser adicionada dentro da tag "<head>" do html ou no final do arquivo após a última tag de "<html>".

```
<script src="meuArquivo.js">
```

O atributo *src* deve ser preenchido com o caminho do arquivo.

Para adicionar mais de um arquivo Javascript, basta duplicar a linha, colocando abaixo o nome do novo arquivo. É importante ressaltar que a leitura é hierárquica, ou seja, caso exista alguma variável que deverá ser lida por outro arquivo, é importante que esta esteja acima no arquivo, para que ela possa já existir no momento da leitura.

Imprimindo texto

Os comandos para impressão de texto do Javascript podem variar de acordo com a necessidade. Pode-se imprimir texto no próprio corpo do HTML, na janela console (output) do navegador ou em pop-ups.

Console

Para acessar o console do navegador deve-se clicar na tecla F12 (Chrome, Firefox ou Internet Explorer). Na janela ou aba que será aberta, procure por "console" ou "output". Para imprimir neste caso, utilizamos as opções a seguir:

Console.log(“Meu texto”)	Imprime um texto.
Console.warn(“Meu texto”)	Imprime um texto cor laranja.
Console.error(“Meu texto”)	Imprime um texto de vermelha.
Console.info(“Meu texto”)	Imprime texto e marca com ícone de informação.

Tela

Para impressão em tela, pode-se utilizar o comando demonstrado abaixo. O texto impresso será colocado no próprio HTML da página na parte superior, todo o conteúdo abaixo deste texto será empurrado para baixo automaticamente.

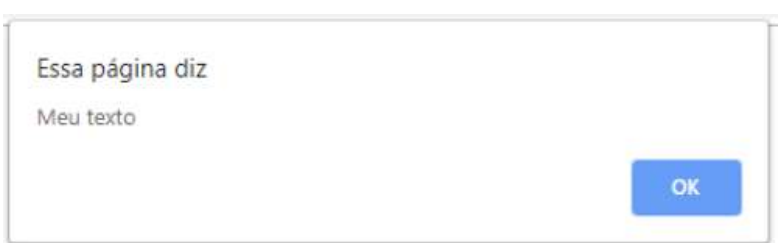
```
document.write(“Meu texto”);
```

Caso se deseje colocar um texto em um lugar específico, devemos utilizar a tag “getElementBy”, que estudaremos mais à frente.

Pop-up

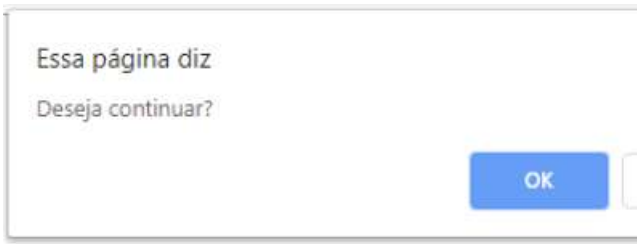
Existe mais de uma forma de pop-up utilizando Javascript. Vejamos abaixo exemplos:

```
alert(“Meu texto”);
```



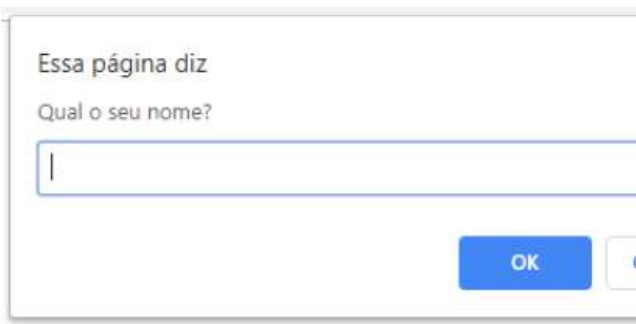
O comando acima exibe uma janela com alerta e um botão de OK.

```
confirm(“Deseja continuar”);
```



O comando acima exibe uma janela com alerta, um botão de Ok e outro de Cancelar.

```
prompt(“Qual o seu nome?”);
```



O comando acima exibe uma janela com alerta, um botão de OK, outro de Cancelar e um campo de texto para digitação.

Para capturarmos o resultado do usuário ao clicar em Ok, Cancelar ou o texto informado, devemos colocar uma variável antes do chamado dos popups. Caso seja clicado em Cancelar, o retorno será nulo.

```
var resultado = Prompt(“Qual o seu nome?”);
```

Veremos mais sobre variáveis e valores nulos na seção Tipos de Dados.

Função

Os códigos citados acima são adicionados dentro do corpo do arquivo Javascript, este que é interpretado no momento da execução da página HTML. Porém, em muitos momentos, gostaríamos que nosso código fosse executado apenas quando alguém fizer alguma ação, como apertar um botão por exemplo. Para isso utilizamos funções.

Funções são blocos de códigos que ficam agrupados e apenas são executados quando chamados. Para criarmos uma função utilizamos a palavra chave "function". Também precisamos definir um nome e utilizar um "()" no final.

```
function MinhaFuncao()
{
    ......
}
```

No nome da função, não podemos utilizar espaços, caracteres especiais ou acentos. É recomendado que os nomes descrevam de forma geral o objetivo que a função tem e estejam sempre flexionados no infinitivo. Caso seja necessário para nomes longos, podemos juntar os nomes com underlines ou escrever tudo junto separando as palavras por letras maiúsculas no início de cada uma. Este último método é chamado de CamelCase.

Veja abaixo alguns exemplos de nomes de funções:

```
function GerarRelatorio()
function BuscarTodosClientes()
function Salvar_Contato()
function Enviar_Email()
```

Chamamos de corpo da função a região onde é escrito o código que será executado. Quanto menor for uma função, melhor. Caso sua função esteja ficando muito longa (mais que 20 linhas), é recomendado dividi-la em duas ou mais funções.

Tenha em mente que uma função é um bloco de código que pode ser acionado a todo momento que você necessite daquela ação, portanto ela não pode ter muita dependência de outros códigos fora da função.

```
function ImprimirNaTela()
{
    document.write("Meu texto");
}
```

Operadores Aritméticos

Operadores Aritméticos são os famosos sinais que utilizamos na matemática para realizar cálculos. No Javascript eles têm a mesma função e nos ajudam na execução de uma lógica ou em realizar um cálculo.

+ (a + b)	Somar	% (a % b)	Resto de uma divisão
- (a – b)	Subtrair	** (a ** b)	Potência
* (a * b)	Multiplicar	= (a = 1)	Igual. Atribui valor para uma variável
/ (a / b)	Dividir		

Parâmetros e retornos de funções

Ao chamar uma função, muitas vezes vamos precisar oferecer algum dado para o código que será executado. Este dado passado é chamado de parâmetro e pode ser utilizado em mais de um caso, se necessário.

```
ImprimirNaTela(“Olá”);

function ImprimirNaTela(msg)
{
document.write(msg);
}
```

No exemplo acima, na chamada do método enviamos o texto “Olá” para a função. A função recebe o texto e armazena na variável “msg”. Quando posteriormente imprimirmos o conteúdo da variável, o resultado será a mensagem “olá” enviada anteriormente.

A variável “msg” estará disponível para uso apenas dentro da função “ImprimirNaTela”, neste caso. Chamamos isso de escopo, que neste caso é interno a esta função.

Tipos de dados

São os formatos de dados que podemos armazenar dentro do nosso código fonte. Durante o desenvolvimento de um projeto, muitas vezes precisamos armazenar informações para acessar em outro momento. Por isso, deixamos tais informações no que chamamos de variáveis. Estas variáveis possuem formatos específicos, que aceitam apenas determinados valores.

Os formatos de dados existem na grande maioria das linguagens de programação e ajudam o computador a entender melhor o código fonte e executá-lo de

forma mais rápida e segura.

Os tipos de dados disponíveis no Javascript são:

- String;
- Número;
- Booleano;
- Array;
- Objeto.

Variáveis e seus tipos

Podemos entender variáveis como espaços dentro da memória do computador, que serão utilizados para armazenar algum dado que definimos. Apesar do conceito ser um pouco complexo, a utilização é simples e muito comum. Variáveis e métodos são os recursos mais utilizados durante o desenvolvimento de um código. Para criar uma variável basta utilizar a palavra chave "var" e logo após definir um nome para ela.

```
var x;
```

A linha de código acima define uma variável de nome "x". Podemos utilizá-la dentro do código porém no momento ela não possui nenhuma informação.

```
var x = "Olá"
```

Neste momento atribuímos um texto à nossa variável "x". A partir daqui também definimos, de forma implícita, que nossa variável usa o tipo de dado "String", pois ela armazena um texto.

```
var y = 1
```

No caso acima, criamos uma nova variável chamada "y" e armazenamos o número 1 dentro dela. Desta forma essa variável assumiu o tipo "Número" ou também chamado de "Inteiro".

```
var y = 1 + 1
```

Como mencionado, variáveis são úteis para o armazenamento de dados para utilização posterior. Agora armazenamos o resultado de um cálculo.

```
var z = true
```

Variáveis booleanas são utilizadas para armazenamento de dois valores apenas, "true" ou "false". Utilizamos este tipo de variável quando trabalhamos com condições lógicas, como por exemplo o "IF", que veremos mais à frente. Também podemos armazenar respostas de perguntas que fazemos ao usuário, como por exemplo "Maior de idade?", "Possui filhos?", "É fumante?".

```
var t = ["Vermelho","Verde","Azul"]
```

Este exemplo é um "Array", com este tipo de estrutura podemos armazenar mais de uma informação em uma mesma variável. Por exemplo, no array "t" armazenamos três nomes de cores. Para capturar os valores posteriormente utilizamos a posição de cada cor, estas que sempre serão iniciadas em 0.

```
t[0]    ->    "Vermelho"
t[1]    ->    "Verde"
t[2]    ->    "Azul"
```

```
console.write(t[1])
```

O código imprimirá a cor "verde".

```
var meuCarro = new Object();
meuCarro.fabricacao = "Ford";
meuCarro.modelo = "Mustang";
meuCarro.ano = 1969;
```

O último exemplo é da criação de um “Objeto”. Esta variável, assim como a array, é utilizada para armazenar mais de uma informação. Porém, neste caso não costumamos armazenar uma lista de valores, ou valores sequenciais. Objetos costumam ser utilizados para armazenar informações que estão relacionadas, como propriedades ou características de algo maior.

No exemplo acima, criamos um carro e colocamos nas suas propriedades outras informações relacionadas.

Para acessar cada uma das informações devemos utilizar o nome da variável e sua propriedade.

```
meuCarro.fabricacao
```

Esta linha imprimirá a informação “Ford”.

Conversão de tipo

Como já vimos, variáveis são utilizadas para armazenar valores que um usuário informou, ou originário de alguma manipulação feita por nós no próprio código, por exemplo, quando somamos dois números e armazenamos o resultado. Vamos ver um exemplo abaixo:

```
var x = 1;
var y = 2;
var resultado = x + y;
```

Neste exemplo utilizamos três variáveis, armazenamos os valores de x e y e então somamos ambos para guardar o resultado. Funcionamento perfeito. Porém, no caso abaixo temos um problema.

```
var x = “1”;
var y = 2;
var resultado = x + y;
```

A variável y é um número, porém a x foi reconhecida como um texto, por isso não podemos somar os dois valores. Uma exigência do Javascript é que para algumas manipulações os dados precisam ser do mesmo tipo, então para seguir com o código acima temos que fazer uma conversão de tipo.

```
var x = “1”;
var xx = parseInt(x);
var y = 2;
var resultado = xx + y;
```

Adicionamos na segunda linha uma conversão para transformar o texto de x em um número e armazená-lo em uma nova variável xx, podendo assim realizar o cálculo. Vejamos também outros exemplos de conversões abaixo:

```
var x = 1;
var xx = String(x);     -> Transforma número em string
```

```
var x = 1;
var xx = parseFloat(x);     -> Transforma número em decimal Ex. 1.0
```

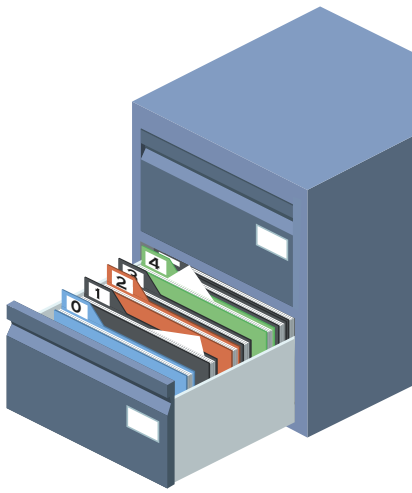
Funções e manipulação de arrays e objetos

Arrays e objetos são variáveis que podemos utilizar quando queremos armazenar uma quantidade maior de dados, arrays no caso de listas ou pilhas, objetos no caso de dados relacionados. Estas duas variáveis também possuem uma gama de funções que nos ajudam a manipular os dados dentro delas.

Imagine que durante o desenvolvimento nós precisemos ordenar, adicionar ou remover dados de dentro de um array ou objeto. Tal manipulação exige formas diferentes de uma simples variável string ou número.

Array

```
var myArray = [“Banana”, “Laranja”, “Maçã”, “Manga”]
```



myArray.pop()	Remove o último elemento
myArray.push(“Abacaxi”)	Adiciona um elemento no final
myArray.shift()	Remove o primeiro elemento
myArray[2]	Retorna o elemento da posição 2 (iniciado em 0)
myArray[2] = “Morango”	Sobrepõe o elemento da posição 2 (iniciado em 0)
myArray.concat(myArray2)	Junta dois arrays em um
myArray.slice(2)	Gera um novo array até a posição 2, demais itens são desconsiderados
myArray.toString()	Retorna todos os itens em forma de texto separado por vírgula

Existem outras funções para manipular arrays, porém esses acima são os mais utilizados. Os itens também podem ser combinados para alcançar outros resultados. A melhor forma de aprender e experimentar cada um deles.

Objetos

```
var pessoa = {nome:”Miguel”, idade:”50”, altura:1.82};
```

{
nome: ”Pedro”
idade: 38,
peso: “57kg”
}

pessoa.nome	Remove o último elemento
pessoa[“nome”]	Adiciona elemento no final
pessoa.nome = “Pedro”	Remove primeiro elemento
pessoa[“nome”] = “Pedro”	Retorna o elemento da posição 2 (iniciado em 0)
pessoa = null	Apaga todas as propriedades do objeto

Também é possível juntar as duas opções de variáveis e ter assim uma variável mais completa, criando um array de objetos. Vamos praticar, crie 3 objetos chamados pessoa1, pessoa2 e pessoa3, similares a variável pessoa acima. Agora atribua valores diferentes para cada um e armazene-os dentro de um array.

```
var meuArray = [pessoa1, pessoa2, pessoa3]
```

Para acessá-los utilize a combinação de ambos novamente, que retorna o nome contido no objeto que está na posição 0.

```
meuArray[0].nome
```

Funções de strings

Variáveis do tipo texto são muito versáteis e podem ser utilizadas em inúmeras ocasiões durante o desenvolvimento. Devido a isso, existem diversas funções que podemos utilizar para manipular seu conteúdo, como por exemplo dividi-lo, capturar o primeiro ou último caractere, contar os caracteres etc.

```
var meuTexto = “Querido computador”
```

meuTexto.length	Retorna o número de caracteres
meuTexto.indexOf(“c”)	Retorna a posição da ocorrência do caractere
meuTexto.slice(0, 7)	Extrai o texto entre 0 e 7
meuTexto.substring(8)	Extrai o texto a partir da posição 8
meuTexto.replace(“computador”, ”notebook”)	Substitui o primeiro texto pelo segundo
meuTexto.toUpperCase() ou toLoweCase()	Transforma todo texto em maiúsculo ou minúsculo
meuTexto.concat(“ super”)	Junta o texto com o existente
meuTexto.trim()	Remove espaços do final e do início do texto

Tipos de eventos

Dentro de um HTML, é necessário adicionar eventos para acionar os códigos Javascript que escrevemos. Geralmente eventos estão vinculados aos botões ou outros campos de formulários, porém é possível adicioná-los a qualquer elemento HTML.

Veja a lista de eventos:

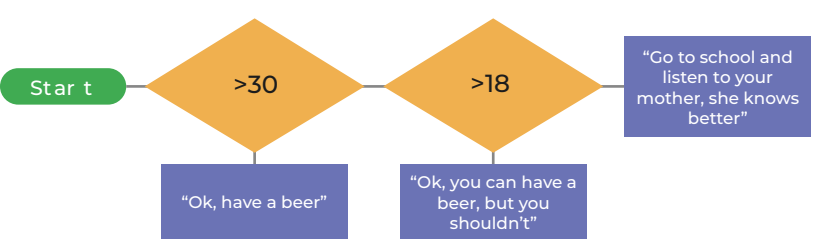
onchange	Ao alterar um elemento
onclick	Ao clicar em um elemento
onmouseover	Ao passar o mouse sobre
onmouseout	Ao tirar o mouse de sobre
onkeydown	Ao clicar em uma tecla do teclado
onload	Ao carregar o elemento HTML

Adicionamos os eventos dentro de um elemento HTML como se fossem uma propriedade da tag, seguido do nome da função Javascript que queremos chamar, dentro de aspas.

```
<button onclick=”Salvar()”>Salvar</button>
```

Eventos são utilizados tanto para chamar métodos que realizam alguma função no nosso código, como também para adicionar efeitos visuais. Podemos, por exemplo, utilizar o evento “onmouseover” para alterar a cor ou o fundo de elemento HTML, causando assim um efeito visual.

Condições utilizando IF e SWITCH CASE



Dentro do Javascript, caso queiramos diferentes ações dependendo do dado a ser enviado pelo usuário, temos que utilizar condições. Existem duas formas de atingir este objetivo, porém cada uma delas tem uma convenção de uso.

IF / ELSE / ELSE IF: São geralmente utilizadas quando temos uma condição que não conseguimos prever todas as possibilidades. Exemplo: caso um número seja maior que 17 realizar x, caso seja menor, realizar y. Neste caso podemos ver que as possibilidades são infinitas.

```
var idade = 20;
if(idade > 17)
  alert(“Maior de idade.”);
else
  alert(“Menor de idade”);
```

SWITCH CASE: São geralmente utilizadas quando as possibilidades são limitadas ou controladas. Exemplo: ações diferentes executadas para cada tipo de moradia.

```
var moradia = “casa”;
switch(moradia)
{
  case “casa”:
    Alert(“Moro em casa”);
  break;
  case “apartamento”:
    Alert(“Moro em apartamento”);
  break;
  case “sobrado”:
    Alert(“Moro em sobrado”);
  break;
  case “cobertura”:
    Alert(“Moro em cobertura”);
  break;
  default:
    Alert(“Nenhuma das opções”);
}
```

Podemos notar que ambas as estruturas podem serem utilizadas para a mesma função, porém a utilização da adequada fará com que escrevemos menos código. Temos também que

sempre considerar como o usuário inserirá a informação que vamos tratar, será um campo livre para digitação ou um campo de escolha? Entendendo quais são as possibilidades de entrada, saberemos qual a melhor forma de tratar uma condição.

Loop utilizando FOR e WHILE

Caso precisemos que alguma ação seja tomada repetidas vezes enquanto uma condição não for atendida, podemos utilizar loops com FOR ou WHILE. Ambas as opções têm o mesmo intuito, porém sua forma de condição pode ser diferente.

FOR: Deve ser especificada qual a condição para que o trecho seja repetido, geralmente um contador é criado, veja abaixo:

```
var contador = 0;
for(contador = 0; contador < 10; contador ++)
{
  alert(“Passo ” + contador);
}
```

O código acima rodará por 10 vezes, somente depois ele seguirá o fluxo do código. Podemos ler a estrutura como: FOR(De; Até; Passo), indicando assim sua condição para fim do fluxo.

Caso desejemos que um código se repita até que uma condição seja alterada, porém não temos certeza de quando isso acontecerá, podemos utilizar o WHILE.

```
var condicao = false;
while (condicao == false)
{
  alert(“Condicao falsa”);
  condicao = true;
}
```

Acima o código será executado apenas uma vez, isto acontece porque a estrutura só será executada caso a variável “condição” seja falsa. Logo na primeira execução, na última linha do código, atribuímos “true” para a variável. Desta forma, na segunda vez que o WHILE verificar sua condição de repetição, ela não será mais satisfeita e o fluxo seguirá normalmente.

CAPÍTULO

05

JQUERY

5.1 Objetivo e uso

A linguagem Javascript é muito extensa e continua sendo atualizada ao longo dos anos com diversos novos recursos, ampliando assim as possibilidades do programador. Dentro destas atualizações, foi criado um novo conceito, chamado de biblioteca. Bibliotecas são códigos Javascripts que são escritos por outros programadores e disponibilizados para uso aberto. Geralmente estas bibliotecas nos trazem formas mais fáceis de realizar algumas ações que caso precisássemos escrever, levaria muito tempo.

Em resumo, bibliotecas são formas mais simples de escrevermos o Javascript e a Jquery é uma das bibliotecas mais populares que vem sendo utilizada e atualizada constantemente. Ela nos auxilia muito, principalmente no desenvolvimento de layouts mais elegantes, com animações e efeitos. Porém, vale lembrar que todo o código utilizado ainda é apenas Javascript.

5.2 Baixando e instalando

Para utilizarmos a Jquery existem duas possibilidades: podemos baixar o código fonte dela e fazer uma referência para nosso projeto utilizá-la, ou podemos colocar um link para consulta neste código fonte, diretamente online. Ambas as possibilidades são gratuitas.

Para realizar o download basta irmos ao site jquery.com e depois adicionar o arquivo dentro da pasta do nosso projeto. Após isso fazemos uma referência a ele dentro do <head> do HTML.

```
<head>
      <script  src=”jquery-3.4.1.min.js”></
script>
</head>
```

Caso optemos pela referência direta online, temos que adicionar o mesmo link acima dentro do <head> do nosso HTML.

```
<head>
    <script  src=”https://ajax.googleapis.
com/ajax/libs/jquery/3.4.1/jquery.min.
js”></script>
</head>
```

Após isso basta iniciar o uso, apenas tenha sempre certeza de que o Jquery está adicionado no mesmo HTML que você está chamando seu código Javascript, para que assim o arquivo .js posso acessar os dados dentro do arquivo Jquery.

5.3 Seletores

Para recuperarmos ou atribuirmos um dado a um elemento HTML, utilizamos os chamados seletores. Eles são a forma de conectarmos o Jquery à tag que será utilizada, de uma maneira similar ao getElementById que vimos no conteúdo Javascript.

Seletor padrão: $(“”)

Sempre utilizamos o código acima como base para fazer qualquer seleção de elemento HTML. Dentro das aspas temos a opção de colocar diretamente qual o elemento queremos, através do nome da tag, id ou classe.

```
<p id="meuId" class="minhaClasse">
Meu parágrafo no html.</p>
```

```
$("p")
$(".meuId")
$("#minhaClasse")
```

Na primeira linha capturamos a frase existente dentro da tag p. Na segunda linha atribuímos um novo texto que irá sobrepor o existente.

5.4 Eventos de interação com mouse

Assim como temos os eventos da forma pura do Javascript, também podemos executá-los via Jquery. Sua sintaxe pode parecer mais complicada no início, porém a facilidade para uso em larga escala é muito maior, além da variedade maior de eventos já prontos para uso.

mousedown()	Ao clicar sobre um elemento
mouseenter()	Ao entrar com a seta do mouse em uma região do elemento
mouseleave()	Ao sair com a seta do mouse de uma região do elemento
mousemove()	Ao mover o mouse dentro da página
mouseout()	Ao sair com a seta do mouse de uma região do elemento. Diferente de muserelease, este evento também considera os elementos filhos
mouseover()	Ao entrar com a seta do mouse em uma região do elemento. Diferente de mouseenter, este evento também considera os elementos filhos
mouseup()	Ao soltar o click sobre um elemento

Para utilizarmos os eventos acima, devemos escrever a estrutura mostrada abaixo. Troque o texto "elemento" pela tag HTML que deseja que o evento atrele ao texto que deseja utilizar.

```
$("elemento").evento(function(){
  alert("Meu evento ocorreu.");
});
```

5.5 Efeitos

O grande charme de utilizar Jquery ao invés de Javascript puro são os recursos de efeitos que temos a disposição. Vejamos alguns deles.

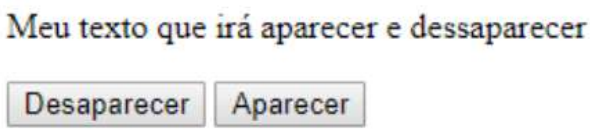
Fade

É o efeito de fazer algo aparecer ou desaparecer na tela. Colocamos este efeito nos elementos HTML e inclusive determinamos um tempo para a execução dele. Os exemplos abaixo adicionam um evento de aparecer e desaparecer a partir do click de um botão. O elemento que está sofrendo a ação é uma tag p do HTML.

```
<p>Meu texto que irá aparecer e desaparecer</p>
<button id="btDesaparecer">Desaparecer</button>
<button id="btAparecer">Aparecer</button>
```

```
$(".btDesaparecer").click(function(){
  $("p").fadeIn();
});
```

```
$(".btAparecer").click(function(){
  $("p").fadeOut();
});
```



Podemos utilizar também os efeitos abaixo na mesma estrutura, os valores de velocidade de execução também podem serem adicionados em qualquer um destes eventos.

.hide()	Esconde o elemento sem utilizar efeito
.show()	Exibe o elemento sem utilizar efeito
.slideDown()	Exibe o item com rolagem para baixo
.slideUp()	Esconde o item com rolagem para cima
.slideToggle()	Esconde o item e exibe com rolagem, engloba os itens Down e Up juntos
.slideToggle(3000)	Executa o mesmo efeito do item acima, porém com 3 segundos de efeito
.slideToggle(“slow”)	Executa o mesmo efeito do item acima, porém de forma lenta. Também disponível na versão fast

CAPÍTULO

06

CSS

6.1 CSS

CSS é uma linguagem que utilizamos para aplicar um layout a uma página HTML. É composta por tags, propriedades e valores. Seu uso está sempre atrelado a um código HTML, pois caso não esteja, não terá utilidade alguma.

O desenvolvimento de código CSS pode ser feito dentro do próprio arquivo HTML ou em um arquivo separado com a extensão ".css". O arquivo, utilizado deve ser referenciado dentro do head do HTML, com uma tag específica. Também pode se utilizar mais de um arquivo, caso se deseje separar o código.

Porque utilizar CSS

Utilizamos CSS principalmente para centralizar tudo que se refere ao estilo de uma página web em um só lugar. Por exemplo, se quisermos definir a fonte do nosso site para tamanho 14, caso nosso site tenha 15 páginas, esta definição feita via HTML teria que ser colocada nas 15 páginas. Se utilizarmos um arquivo CSS, todas as páginas podem referenciar este mesmo arquivo. Ao alterar os códigos do arquivo, todas as páginas são afetadas simultaneamente. Esta abordagem ajuda muito durante o desenvolvimento e manutenção do dia a dia.

Adicionando CSS a uma página HTML

Para referenciar um arquivo CSS dentro de um arquivo HTML, temos que adicionar uma tag específica para esta função. Esta tag possui uma propriedade "href", que deve especificar o nome do arquivo de estilo que criamos. É muito importante colocar o caminho correto do arquivo caso ele não esteja no mesmo nível que o arquivo HTML.

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

Desta forma, todo código CSS que adicionarmos no arquivo será aplicado ao código HTML que aparecerá na página.

Seletores

Chamamos de seletores os métodos ou meios que temos para selecionar um elemento HTML no qual queremos aplicar um estilo. Atualmente podemos utilizar dentro do CSS três formas de seleção, via classe, id ou tipo da tag. A utilização de seletores é primordial para o uso do CSS. Sem eles não conseguimos dizer para o nosso CSS em qual elemento HTML deve ser aplicado as definições que estamos incluindo.

.	Seletor para classe
#	Seletor para id
<tag>	Não se deve colocar os sinais "<" e ">" que colocamos no HTML ao utilizarmos a tag, exemplos são: p, div, span

Para entender melhor o uso dos seletores acima, devemos ver também o uso de classes, já que ambos são utilizados em conjunto.

Classe

É um agrupamento de código CSS, que possui um nome e geralmente se refere a um mesmo trecho ou funcionalidade. Por exemplo, quando criamos um botão em uma página, temos que definir seu tamanho, fonte, cor, borda etc. Todos estes itens relacionados ao botão devem ficar em uma mesma classe, desta forma ao adicionar esta classe no HTML do botão, todo o estilo é aplicado imediatamente.

```
Botao{
background-color: Green;
font-size: 16px;
color: white;
}
```

Esta é a estrutura de uma classe CSS. Demos o nome de "botao" e definimos uma cor de fundo, tamanho, texto e cor do texto. Temos que definir agora qual seletor iremos utilizar, vejamos as três possibilidades:

```
.Botao{
background-color: Green;
font-size: 16px;
color: white;
}
```

```
#Botao{
background-color: Green;
font-size: 16px;
color: white;
}
```

```
button{
background-color: Green;
font-size: 16px;
color: white;
}
```

O primeiro seletor (".") usamos para selecionar através da classe, o segundo ("#") utilizamos para selecionar através do id, o terceiro é através do tipo da tag. Neste último caso o estilo é aplicado a todas as tags de botão dentro do HTML, ou seja, não é recomendado caso não se tenha intenção de aplicar o layout a todos os botões. Vejamos como deve ficar o HTML do botão.

```
Via tipo da tag
<button>Ok</button>
```

```
Via id
<button id="Botao">Ok</button>
```

```
Via classe
<button class="Botao">Ok</button>
```

Nos próximos exemplos que mostraremos nesta apostila, sempre utilizaremos a opção de classe.

Formatando textos

Agora que já sabemos montar uma classe e utilizar um seletor para ligarmos nosso código a um elemento HTML, vamos ver algumas das muitas opções de propriedade que o CSS nos oferece. Entre as mais comuns estão as propriedades para formatação de texto, veja abaixo:

color: blue	Define a cor do texto
text-align: right	Define o alinhamento, entre as opções estão: right, left center ou justify
text-decoration: none	Define que o texto será sublinhado, com um traço cortando no meio ou em cima. A opção None é utilizada quando nenhum traço é utilizado
text-transform: uppercase	Define o texto como maiúsculo. A opção Lowercase é utilizada para texto minúsculo, e a opção Capitalize para primeira letra maiúsculas e demais minúsculas
letter-spacing: 3px	Define o espaçamento entre as letras
line-height: 1.5	Define o espaçamento entre as linhas
word-spacing: 10px	Define o espaçamento entre palavras
text-shadow: 1px red	Define a sombra de um texto com as variáveis de posição horizontal, vertical e cor

No final, basta agruparmos as que desejamos dentro de uma classe, da seguinte maneira:

```
.Titulo{
    color: black;
    font-size: 15px;
    text-align: center;
    letter-spacing: 0.8;
}
```

```
.SubTitulo{
    color: Silver;
    font-size: 13px;
    text-align: left;
    letter-spacing: 0.8;
}
```

Note que para separar uma propriedade da outra dentro de uma classe utilizamos ponto e vírgula.

Formatando conteúdos

O CSS também oferece muitas propriedades para formatação de blocos HTML, ou seja, regiões que contêm diversos elementos HTML dentro. Geralmente estes blocos são formados por div, tabelas (table) ou listas (ul ou ol). Tais elementos, após estilizados, podem gerar estrutura super avançadas, como menus, galerias, áreas para notícias, entre outros.

DIV

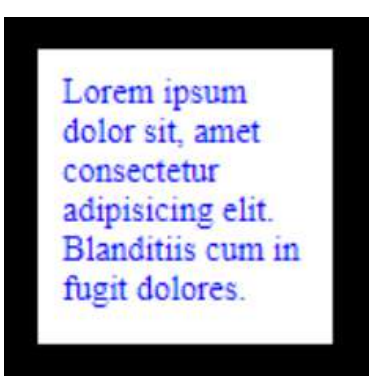
As divs são blocos HTML que sem formatação de layout, que não afetam em nada uma interface. São transparentes, sem borda e geralmente utilizados para agrupar blocos do nosso HTML, porém ao formatá-las com algum estilo podemos mostrar o seu real formato na tela. Ao colocarmos uma borda ou fundo, por exemplo, veremos que ela possui o formato de uma caixa. Podemos estilizar o texto interno, criar espaçamentos, adicionar imagens e logo teremos um bloco bastante atraente para incluir em uma página.

Diferente das propriedades utilizadas em texto, para este tipo de elemento usamos as seguintes:

width: “300px”	Define a largura
height: 200px	Define a altura
border: 15px solid green	Define espessura, tipo e cor da borda
padding: 5px	Define o espaço entre a borda e o conteúdo interno
margin: 10px	Define o espaço entre a borda e o conteúdo externo
background-color: silver	Define a cor para o fundo

Os itens acima servem para a definição do layout da div. Para formatar o conteúdo da div podemos utilizar as tags de formatação de texto que já mostramos. Caso utilizemos no código alguma propriedade que não pode ser aplicada em algum elemento HTML, ela simplesmente será ignorada.

```
div{
    width: 100px;
    height: 100px;
    border: 15px solid black;
    padding: 10px;
    font-size: 14px;
    color: Blue;
}
```



A formatação acima montará uma caixa na interface e colocará também os textos internos para tamanho 14 com cor azul. O próprio HTML sabe quais propriedades ele deve aplicar na agrupadora div e quais aplicar nos textos.

Tabelas

Como vimos no HTML, as tabelas são estruturas muito versáteis e nos ajudam a organizar conteúdo dentro de uma página. Elas são compostas por linhas e colunas, com opção de linha para título e para rodapé. Para cada uma destas partes podemos ter uma formatação diferente, vamos considerar a estrutura abaixo:

```
<table>
    <thead>
      <th>Título 1</th>
      <th>Título 2</th>
    </thead>
    <tbody>
            <tr>
                    <td>Texto 1</td>
                    <td>Texto 2</td>
            </tr>
    </tbody>
</table>
```

Para definirmos as bordas de cada linha e coluna adicionamos uma classe a uma tag table, abaixo definimos seu tamanho para ocupar toda a largura da página.

```
table {
    border: 3px solid black;
    width: 100%;
}
```

Definimos o th e td para o texto alinhar à esquerda e seu espaçamento com a linha da borda ser de 15px.

```
th, td {
    border: 1px solid black;
padding: 15px;
text-align: left;
}
```

Titulo 1	Titulo 2
Texto 1	Texto 2

Listas

Uma lista, seja ela ordenada ou não ordenada, é composta por um marcador e um texto. Logo A formatação que mais prevalece é a feita nos próprios textos, porém podemos utilizar CSS para criar uma classe e definir uma estrutura básica.

```
ul {
            list-style-type: square;
            margin: 10px;
    padding: 5px;
    font-family: Verdana;
}
```

- Item 1
- Item 2
- Item 3

Para uma lista ordenada, podemos utilizar como marcador uma imagem própria, lembrando que a imagem deve estar na mesma pasta do seu arquivo CSS e com o tamanho apropriado.

```
ol {
    list-style-image: url('bulet.gif');
}
```

- Coffee
- Tea
- Coca Cola

Imagens

Existem diversas formas de se utilizar imagens, elas podem fazer parte de uma galeria, aparecer no corpo de uma notícia ou até mesmo como plano de fundo. Para utilizarmos uma imagem é importante dispormos dela no tamanho correto que precisamos e dentro de uma pasta do seu projeto.

Primeiros vamos ver como adicionar uma imagem como plano de fundo de uma div via CSS. Repare que utilizamos o caminho da imagem de forma relativa, ou seja, a partir da pasta do CSS, voltamos um nível e acessamos a pasta img. Caso a imagem esteja no mesmo nível do arquivo CSS, basta colocar o nome da imagem no código.

```
    div{
        background-image:    url("../img/
imagem.jpg");
        width: 50px;
            height: 40px;
            background-repeat: repeat-x;
    }
```

Também definimos uma largura e altura para a imagem, porém isto não é recomendado, pois deformará a resolução da mesma. O ideal é a imagem já estar redimensionada no tamanho que precisamos. A propriedade "background-repeat" diz para a imagem se repetir caso ela não cubra todo o fundo da div. Definimos neste exemplo apenas o eixo x, também podemos utilizar a opção "no-repeat" ou "cover" na imagem, se não desejamos que ela se repita.

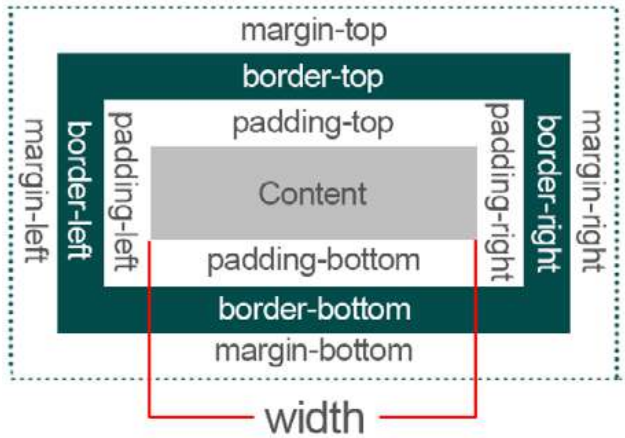
Agora para adicionarmos uma formatação CSS a uma tag image do HTML, podemos seguir os itens abaixo.

```
Image{
    width: 50px;
    height: 40px;
    float: left;
    border: 1px solid #ccc;
}
```

Neste caso a URL da imagem fica definida na própria tag HTML. No CSS apenas definimos seu tamanho, float (sua posição na tela) e borda. O mesmo pode ser feito caso utilizemos uma imagem como ícone.

Bordas, margens e padding

Estes são recursos muito úteis para a formatação de uma página HTML. Aplicamos constantemente essas propriedades em todo tipo de conteúdo, principalmente em agrupadores. Existe uma certa similaridade entre essas propriedades que podem fazer com que as utilizemos de forma inapropriada. Todas elas têm o objetivo de gerar um espaçamento entre dois pontos, porém com características diferentes. Na imagem abaixo podemos ver claramente a que compete cada propriedade.



Padding: Refere-se ao espaçamento entre conteúdo e borda.

Border: Refere-se a linha da borda.

Margin: Refere-se ao espaçamento entre a borda e o conteúdo externo.

Para cada um dos itens acima, ainda temos a opção de segmentar por direita, esquerda, topo e base. Ou seja, é necessário pensar com calma antes de ir aplicando estas propriedades em todo lugar, o seu código pode acabar ficando confuso e tendo aquele ligeiro desalinhamento que nunca sabemos de onde está vindo. Outra característica desta propriedade é que podemos utilizá-las em tabelas, listas, divs, imagens, formulários, entre outros. Apenas em textos não conseguimos utilizá-las.

Formatando formulários

Um formulário é composto basicamente por inputs e buttons. Não podemos adicionar formatações diretamente para os inputs, pois isso afetaria todos os campos, a não ser que este seja o objetivo. Porém cada campo pode requerer alguma formatação específica de

tamanho, altura ou fonte, para isso podemos utilizar uma propriedade chamada type, onde especificamos para quais tipos de input nossa classe irá atender.

```
input[type=text], select {
  width: 100%;
  padding: 12px 20px;
  margin: 8px 0px;
  border: 1px solid #ccc;
  border-radius: 4px;
}
```

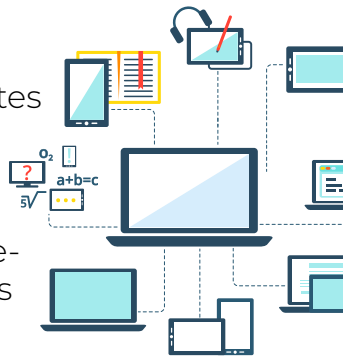
```
input[type=submit] {
  width: 100%;
  background-color: #4CAF50;
  color: white;
  padding: 14px 20px;
  margin: 8px 0px;
  border: none;
  border-radius: 4px;
}
```

```
input[type=submit]:hover {
        background-color: #45a049;
}
```

```
div {
  border-radius: 5px;
  background-color: #f2f2f2;
  padding: 20px;
}
```

Neste layout podemos ver algumas variações e novas propriedades que ainda não utilizamos.

Media Query



Este é um recurso muito importante quando trabalhamos com sites que devem ser visualizados em diversos devices, como mobile, tablet, desktop ou notebook. Com o Media Query aplicamos algumas regras ao CSS dizendo quando um bloco de código deve ser exibido ou não. Apesar de CSS não ser uma linguagem de programação, este é um recurso que nos permite criar condições semelhantes às que podemos criar com a programação.

```
@media not|only mediatype and (expressions) {
    CSS-Code;
}
```

Esta é a estrutura padrão, ela se parece muito com uma classe CSS, porém ela sempre inicia com "@media". Logo após ela apresenta uma condicional, tipo de media e uma expressão. Dentro dos parênteses colocamos nosso bloco de CSS normalmente, ele será executado apenas quando a condição colocada na primeira linha for satisfeita.

Para mediatype temos as seguintes opções:

screen: CSS será aplicado no conteúdo da tela;
print: CSS será aplicado apenas quando a tela for impressa;
speech: CSS será aplicado apenas para software de leitura de tela (acessibilidade);
All: CSS será aplicado para todas as opções listadas acima.

Além dos itens acima, para o CSS realmente ser aplicado a condição dentro dos parênteses também deve ser satisfeita. A condição abaixo é relacionada ao tamanho da página.

```
(min-width: 480px): Css é executado quando o comprimento da tela for menor que 480.
(min-width: 1024px): Css é executado quando o comprimento da tela for menor que 1024.
```

Tendo isso, vejamos o código completo:

```
@media screen and (min-width: 600px) {
  body {
    background-color: lightgreen;
  }
}
```

O código acima aplicará o fundo da página como verde claro quando a resolução da tela for menor que 600px. Para testar, abra o navegador, vá diminuindo com o mouse seu tamanho até que chegue ao tamanho mínimo, a cor do fundo deverá mudar automaticamente.

Da mesma forma podemos aplicar esta regra em todo o site, fazendo com que ele mude seu layout ou esconda itens quando a resolução for menor, como acontece em tablets ou na versão mobile.

Formatando itens para impressão

Dentre as opções que temos de impressão, podemos por exemplo utilizar a opção print do Media Query para aplicar uma imagem de background caso alguém imprima a página web do site, como uma marca d'água da empresa. Também é possível esconder um menu ou outras partes de um site para que a impressão foque apenas em um texto específico.

```
@media print {
  body {
     background-image:   url("logo-empresa.
jpg";
  }
  .Esconder
          {
          Display: none;
}
}
```

A classe acima irá aplicar uma imagem de fundo na tag body apenas quando a página for impressa. Como a tag body corresponde a toda a página, logo o plano de fundo cobrirá toda impressão. A segunda classe, "esconder", será aplicada apenas aos elementos que tiverem esta classe, por exemplo: <p class="Esconder">. Por último, a propriedade display faz referência à exibição de um elemento, se o valor for none, o item será escondido durante a impressão.

Observe abaixo este conteúdo do site no navegador:

- Google
- Uol
- Globo
- Outros

Perferendis earum similique iusto

Lorem ipsum dolor sit amet consectetur adipisicing elit. Perferendis earum similique iusto exercitationem hic eius aperiam deleniti! Sed maiores, officiis deserunt ipsa similique labore praesentium, quisquam fugiat aspernatur impedit unde.

Agora vejemos o conteúdo do site na impressão. Podemos observar que a lista com bullets foi escondida e uma imagem foi aplicada ao fundo.

Páginas responsivas

Este conceito se refere a um site que se adapta ao tamanho de exibição de uma tela, ou seja, ao visualizar uma página em um desktop, notebook, tablet ou mobile, o código tomará as devidas ações para redimensionar o conteúdo e permitir uma navegação normal para o usuário, tudo isso é feito utilizando o mesmo código fonte. Para que isso seja possível, é necessário desenvolver uma página já tendo este objetivo desde o início, pois deve-se seguir algumas regras para atender esta demanda.

```
t="width=device-width, initial-scale=1.0">
```

Esta linha é o primeiro item que deve ser adicionado a uma página responsiva. Ela dirá ao browser como lidar com a dimensão da página.

Outros pontos importantes são:

1. Sempre colocar o CSS no arquivo .css, a parte do código HTML. Códigos inseridos diretamente dentro das tags HTML irão dificultar o trabalho.

2. Usar para as opções width os valores em porcentagem.

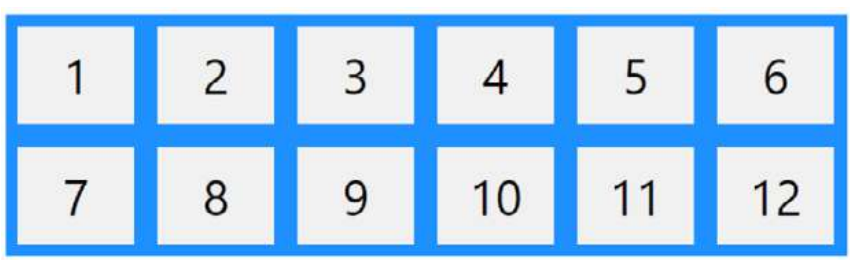
3. Para textos, utilizar a escala vw ao invés de px.

4. Usar Media Queries para quebrar as colunas de informações, esconder ou exibir algum conteúdo.

Seguindo os passos acima já é possível montar um site responsivo. Não é necessário criar nada novo, somente criar da forma correta. Existem algumas formas populares de facilitar a montagem de um site responsivo. Geralmente os chamados frameworks são muito bem-vindos, entre os mais conhecidos estão o Bootstrap e o W3.css.

Flex-box

Este é um recurso do CSS que permite a criação de estruturas responsivas de forma mais simplificada. A Flex-box é conhecida também como caixas flexíveis ou flutuantes, isso devido a forma como o conteúdo é posicionado na página.



É possível perceber pela estrutura acima que os conteúdos são colocados dentro de caixas (cinza), que são enfileiradas até uma determinada quebra de linha, tudo isso dentro de uma caixa maior (azul). O seu tamanho, layout, e número de caixas por linha são definidos pelo próprio desenvolvedor.

```
<div class="flex-container">
    <div>1</div>
    <div>2</div>
    <div>3</div>
</div>

.flex-container {
    display: flex;
}
```

No bloco principal atribuímos uma classe chamada flex-container e nela definimos a propriedade principal display:flex. Para esta classe ainda podemos considerar as propriedades abaixo:

flex-direction	Especifica a direção das caixas
flex-wrap	Define se as caixas quebrarão de linha
justify-content	Define o alinhamento das caixas horizontalmente
align-items	Define o alinhamento das caixas verticalmente
align-content	Define o alinhamento e o espaçamento entre as linhas onde estão as caixas

Também precisamos nos preocupar com as propriedades de cada caixa interna, não somente com a caixa maior em volta. Chamamos estas caixas de caixas filhas e elas podem ter as seguintes propriedades:

order	Especifica a ordem da caixa
flex-grow	Define o quanto uma caixa pode expandir
flex-shrink	Define o quanto uma caixa pode encolher
flex-basis	Define o tamanho inicial de uma caixa em px
align-self	Alinha verticalmente uma caixa específica

Os itens acima devem ser especificados dentro de cada div que corresponde a uma caixa. Vejamos o exemplo:

```
<div class="flex-container">
<div style="order:4">1</div>
<div style="flex-grow:1 ; order:3">2</div>
<div style="order:2">3</div>
<div style="order:1">4</div>
</div>
```

Sobrescrevendo e complementando estilos

Após vermos tantos estilos possíveis para aplicar a uma página, precisamos entender como é seu carregamento dentro da página HTML e como fazemos para sobrepor alguma prosperidade que já vem definida, ou pelo navegador ou por algum framework que estejamos utilizando.

O CSS é uma linguagem cascateada, ou seja, a execução que vier após irá sempre sobrepor a anterior.

Por exemplo se adicionarmos dois arquivos CSS com a tag abaixo:

```
<link rel="stylesheet" type="text/css" href="estilo1.css">
```

```
<link rel="stylesheet" type="text/css" href="estilo2.css">
```

O arquivo estilo1 será executado e aplicará todos as propriedades contidas nele, porém o arquivo número 2 também será executado. Caso haja algo no segundo arquivo que se refere ao mesmo elemento HTML formatado pelo primeiro arquivo, este será complementado ou sobreposto.

<button class=”btSalvar”>Salvar<button>

```
estilo1.css
.btSalvar{
        background-color: green;
        font-color: white;
        font-size: 15px;
        border-style: 2px solid darkgreen;
}
```

```
estilo2.css
.btSalvar{
        background-color: blue;
        font-color: white;
        margin: 10px;
}
```

No exemplo acima os dois arquivos possuem a mesma classe e ambas serão executadas, primeiro a número 1 e logo em seguida a número 2. A propriedade background-color será sobreposta pela segunda classe e será azul. A font-color permanecerá a mesma pois é igual em ambas, já margin será adicionada pois não existia na primeira classe. Tudo que existe na primeira e não é referenciado na segunda, será mantido.

```
Resultado:
.btSalvar{
        background-color: blue;
        font-color: white;
        font-size: 15px;
        border-style: 2px solid darkgreen;
        margin: 10px;
}
```

CAPÍTULO

07

BOOTSTRAP

7.1 Objetivo e uso

Bootstrap é um framework que faz a junção entre HTML, CSS e Javascript, para criar estruturas mais dinâmicas, responsivas e de fácil entendimento. Seguindo o conceito de mobile-first, o site é pensado primeiramente como um site mobile. A versão web é tratada como um complemento da versão mobile, porém isto não significa que ela seja menos importante ou tenha menos funcionalidades.

O Bootstrap também é gratuito e possui uma comunidade bem ativa. Seu site contém toda a documentação de uso, com exemplos e códigos já prontos para implementação. Assim como outros frameworks, ele pode ser utilizado diretamente online ou através do download dos arquivos e inclusão no nosso projeto.

7.2 Baixando e instalando

Para baixar o Bootstrap, acesse o seu site e faça o download da versão desejada, um arquivo compactado com uma pasta CSS e outra JS. Coloque as duas pastas dentro do seu projeto.

Link para download:
https://getbootstrap.com/docs/4.3/getting-started/introduction/

Após adicionar os arquivos ao seu projeto, basta referenciá-los no head do seu arquivo HTML. Todos os arquivos HTML que forem utilizar este recurso devem ter as referências no head.

Exemplo de como referenciar o arquivo baixado:

```
<link rel="stylesheet" href="/css/bootstrap.min.css">
<script src="/js/bootstrap.min.js"></script>
```

Exemplo de como referenciar o arquivo diretamente online:

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
```

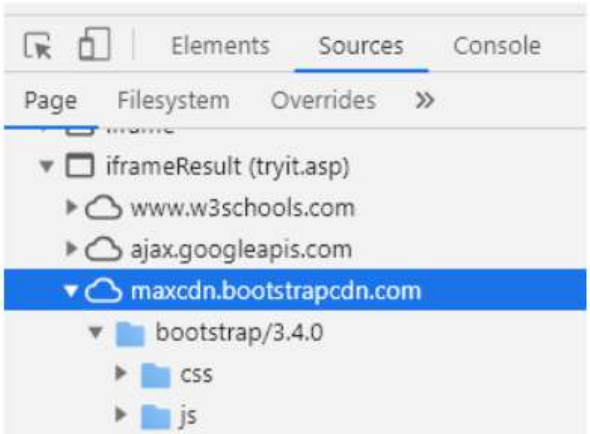
```
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
```

Também é importante adicionar a linha abaixo para garantir a execução responsiva:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

Tenha certeza do caminho dos arquivos do Bootstrap dentro do seu projeto. Uma dica é: após carregar sua página, clique F12 e veja na aba source do seu navegador

para conferir os arquivos externos adicionados.



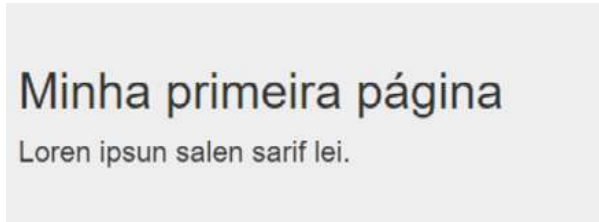
7.3 Painel

Tendo tudo instalado, para começarmos a utilizar o Bootstrap, basta criarmos elementos HTML com as classes pré definidas pelo framework.

```
<div class=”container”>
    <div class=”jumbotron”>
        <h1>Minha primeira página</h1>
        <p>Loren ipsun salen sarif lei.</p>
    </div>
</div>
```

As classes container e jumbotron, quando utilizadas em conjunto, criam um painel para inserção de texto ou imagens. O objetivo não é que este painel apenas se mantenha, mas que possamos complementá-lo e customizá-lo de acordo com o que precisamos. O ponto importante é que já é um painel responsivo e se adapta para outros devices automaticamente, além de conter bordas arredondadas, espaçamentos, cores e fontes pré-definidas.

Ao customizá-lo podemos criar outros arquivos CSS e adicionar uma classe com o mesmo nome, porém com nossas propriedades, que irão sobrepor ou complementar as propriedades padrão do Bootstrap.



7.4 Tipografia

Existem diversas predefinições de textos que podemos utilizar e agilizar nosso trabalho com Front-End. As tags já conhecidas do HTML recebem um layout mais elegante quando utilizamos o Bootstrap. Basta adicionar qualquer uma delas no código e seu estilo já será considerado, não havendo a necessidade de adicionar classes neste caso.

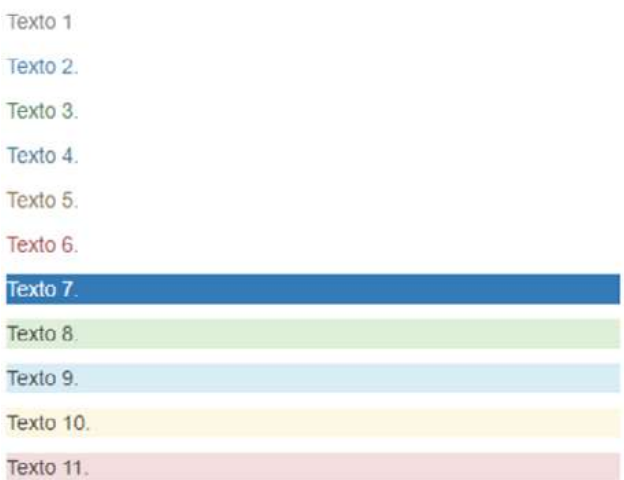
```
<div class=”container”>
        <h1>h1 Título 1</h1>
        <h2>h2 Título 2</h2>
        <h3>h3 Título 3</h3>
        <h4>h4 Título 4</h4>
        <h5>h5 Título 5</h5>
        <h6>h6 Título 6</h6>
</div>
```

Outras tags:

<small>	<code>
<mark>	<kbd>
<abbr>	<pre>

Também temos definições existentes para textos com destaques. Estas requerem adicionar os itens abaixo como classes em uma tag <p>.

.text-muted	.text-primary
.text-info	.text-warning
.text-danger	.bg-primary
.bg-success	.bg-info
.bg-warning	.bg-darger



7.5 Listas

Podemos utilizar toda a versatilidade de uma lista para montar menus. É possível adicionar links, cores e efeitos de contato com o mouse em uma lista, com a estrutura HTML padrão. Para isto, devemos adicionar as classes disponíveis pelo Bootstrap.

```
<ul class="list-group">
    <li class="list-group-item">Item 1</li>
    <li class="list-group-item">Item 2</li>
    <li class="list-group-item">Item 3</li>
</ul>
```

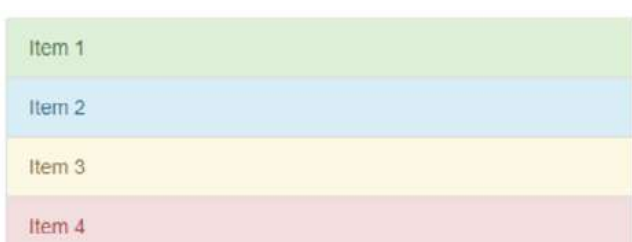
A classe acima cria uma borda com espaçamento, dando a impressão de cada linha ser uma caixa retangular. Também pode ser adicionado no texto de cada linha a tag <a>, fazendo assim com que o texto seja um link.

O trecho abaixo foi utilizado para adicionar o número em destaque no final de cada linha. A classe badge é a responsável pela montagem desta formatação.

```
<span class="badge">12</span>
```

Outras formatações podem ser adicionadas com as classes abaixo, basta adicionar na linha <li>.

list-group-item-success	list-group-item-danger
list-group-item-info	disabled
list-group-item-warning	active



7.6 Tabelas

Similar às listas, também podemos utilizar o Bootstrap para formatar tabelas de forma elegante, com diversas classes já prontas que podem dar um estilo bem harmonioso. Basta adicionar cada uma delas e ver o resultado. A estrutura do HTML se mantém a mesma.

Nome	Cidade	Estado
João	São Paulo	SP
Maria	Rio de Janeiro	RJ
Alicia	Belo Horizonte	MG
Pedro	Salvador	BA
Fernando	Aracaju	SE
Adão	Fortaleza	Ceará

Vamos verificar o código:

```
<table class="table">
  <thead>
    <tr>
      <th>Nome</th>
      <th>Cidade</th>
      <th>Estado</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>João</td>
      <td>São Paulo</td>
      <td>SP</td>
    </tr>
    <tr class="success">
      <td>Maria</td>
      <td>Rio de Janeiro</td>
      <td>RJ</td>
    </tr>
    <tr class="danger">
      <td>Alicia</td>
      <td>Belo Horizonte</td>
      <td>MG</td>
    </tr>
    <tr class="info">
      <td>Pedro</td>
      <td>Salvador</td>
      <td>BA</td>
    </tr>
    <tr class="warning">
      <td>Fernando</td>
      <td>Aracaju</td>
      <td>SE</td>
    </tr>
    <tr class="active">
      <td>Adão</td>
      <td>Fortaleza</td>
      <td>Ceará</td>
    </tr>
  </tbody>
</table>
```

As classes foram destacadas em vermelho. Outras formatações do Bootstrap são aplicadas diretamente nas tags table, thead, tbody, td, th e tr. Temos que lembrar que apesar de ser prático aplicar formatações diretamente a uma tag, isto pode gerar problemas caso desejemos utilizar a tag mais de uma vez na mesma página. No exemplo acima, se tivermos duas tabelas, ambas terão obrigatoriamente o mesmo layout.

7.7 Formulário

Pode se dizer que as classes existentes para formulários são as que mais se destacam, já que aplicam uma formatação bem avançada nos campos e deixam o visual bem atrativo. Campos de formulários são bem difíceis de se posicionarem de forma uniforme, já que parte de sua renderização fica a cargo do navegador.

O primeiro ponto é definir a classe para agrupamento dos campos utilizados no formulário. Lembramos que todos os elementos de um HTML devem estar dentro da tag <form>.

.form-horizontal	Alinha o label e o input lado a lado
.form-control	Alinha o label e o input um em embaixo do outro e deixa eles responsivos
.form-control-feedback	Adiciona layout de erro ou sucesso em um campo input. Deve ser adicionado dentro de uma tag
.form-group	Alinha o label e o input um embaixo do outro

```
<form class="form-horizontal">
        <div class="form-group">
                <label>Email:</label>
                <input type="email" class="form-control" placeholder="Enter email">
        </div>
        <div class="form-group">
                <label>Senha:</label>
                <input type="password" class="form-control" placeholder="Enter password">
                <span class="glyphicon glyphicon-ok form-control-feedback"></span>
        </div>
</form>
```

Para o caso acima, as classes glyphicon e glyphicon-ok são referente ao ícone que será exibido no input em questão. Em resumo, podemos dizer que se deve:

- Escrever os textos como labels e divs de agrupamento, utilizando a classe form--group.

- Escrever os controles de input, textarea e select utilizando a classe form-control.

7.8 Bootstrap Grid

O sistema de grid do Bootstrap é justamente o que permite a responsividade da página. São 12 colunas que ocupam toda a largura da página e podem ser agrupadas ou separadas, para garantir o enquadramento automática em telas de tamanho diferentes.

Observando a imagem percebemos que podemos dividir as sessões de uma página através dessas colunas. Por exemplo, podemos definir que em um monitor grande a tela utilize 3 colunas, já em uma tela menor, duas colunas, e finalmente um mobile, apenas uma coluna. Para tais definições, devemos utilizar as classes abaixo:

xs: para telas menores que 768px.
sm: para telas iguais ou maiores que 768px.
md: para telas iguais ou maiores que 992px.
lg: para telas iguais ou maiores que 1200px.

Estas classes devem ser adicionadas nas divs que representam cada um dos blocos na imagem acima. Para que a div saiba se comportar em cada tipo de tamanho de tela, é necessário adicionar todas as classes e definir quantas colunas serão utilizadas para cada uma delas. Devemos manter em mente sempre a proporção de 12.

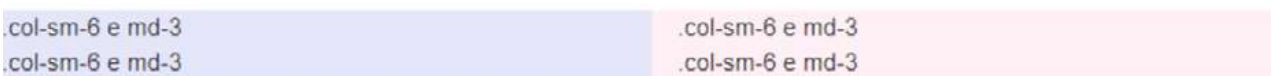
No código abaixo estamos dizendo que existem 3 blocos e cada um deles ocupa 4 partes de 12, veja o resultado:

```
<div class="row">
        <div class="col-sm-4">.col-sm-4</div>
        <div class="col-sm-4">.col-sm-4</div>
        <div class="col-sm-4">.col-sm-4</div>
</div>
```

No caso acima definimos o tamanho dos blocos apenas para a classe sm, ou seja, a tela terá 3 blocos lado a lado, mas apenas se for maior que 768px. Caso desejemos que o número de colunas mude em outros tamanhos de tela, temos que adicionar também as demais classes.

Tela grande

Tela pequena



Para que os exemplos acima funcionem, também são necessárias mais duas outras classes no mesmo contexto. As div para cada bloco devem estar dentro de uma outra div com a classe row. Essa div com row deve estar dentro de mais uma div, com a classe container.

```
<div class="container">
    <div class="row">
            <div class="col-*-*"></div>
            <div class="col-*-*"></div>
    </div>
            <div class="row">
                    <div class="col-*-*"></div>
                    <div class="col-*-*"></div>
                    <div class="col-*-*"></div>
    </div>
    <div class="row">
            ...
    </div>
</div>
```

Com esta estrutura básica para o funcionamento, pode-se adicionar quantas linhas e blocos forem necessários.

Textos

Design
Hugo Zambini
Ricardo Souza

Vetores
freepik.com