

Índice

- [Introdução ao HTML](https://github.com/Webschool-io/s4girls/blob/master/material-didatico/apostila.md#introdu%C3%A7%C3%A3o-ao-html) (<https://github.com/Webschool-io/s4girls/blob/master/material-didatico/apostila.md#introdu%C3%A7%C3%A3o-ao-html>)
- [Introdução à CSS](https://github.com/Webschool-io/s4girls/blob/master/material-didatico/apostila.md#introdu%C3%A7%C3%A3o-%C3%A0-css) (<https://github.com/Webschool-io/s4girls/blob/master/material-didatico/apostila.md#introdu%C3%A7%C3%A3o-%C3%A0-css>)
- [Introdução ao Javascript](https://github.com/Webschool-io/s4girls/blob/master/material-didatico/apostila.md#introdu%C3%A7%C3%A3o-%C3%A0-javascript) (<https://github.com/Webschool-io/s4girls/blob/master/material-didatico/apostila.md#introdu%C3%A7%C3%A3o-%C3%A0-javascript>)
- [Introdução à Estrutura de Dados](https://github.com/Webschool-io/s4girls/blob/master/material-didatico/apostila.md#introdu%C3%A7%C3%A3o-%C3%A0-estrutura-de-dados) (<https://github.com/Webschool-io/s4girls/blob/master/material-didatico/apostila.md#introdu%C3%A7%C3%A3o-%C3%A0-estrutura-de-dados>)
- [Introdução à Lógica](https://github.com/Webschool-io/s4girls/blob/master/material-didatico/apostila.md#introdu%C3%A7%C3%A3o-%C3%A0-l%C3%B3gica) (<https://github.com/Webschool-io/s4girls/blob/master/material-didatico/apostila.md#introdu%C3%A7%C3%A3o-%C3%A0-l%C3%B3gica>)
- [Funções](https://github.com/Webschool-io/s4girls/blob/master/material-didatico/apostila.md#fun%C3%A7%C3%A3o) (<https://github.com/Webschool-io/s4girls/blob/master/material-didatico/apostila.md#fun%C3%A7%C3%A3o>)
- [Objetos](https://github.com/Webschool-io/s4girls/blob/master/material-didatico/apostila.md#objetos) (<https://github.com/Webschool-io/s4girls/blob/master/material-didatico/apostila.md#objetos>)

<div style="page-break-after: always;"></div>

Introdução ao HTML

Linguagem de marcação

HTML é uma **linguagem de marcação**. Uma linguagem de marcação, no sentido em que se relaciona com os navegadores, é uma linguagem com uma sintaxe específica que fornece instruções ao navegador sobre como exibir uma página. A HTML distingue e separa o "conteúdo" (palavras, imagens, áudio, vídeo, e assim por diante) de sua "forma de apresentação" (as instruções sobre como determinado tipo de conteúdo deve ser exibido).

[Developer Mozilla Fundation](https://developer.mozilla.org/pt-BR/docs/HTML/Introduction) (<https://developer.mozilla.org/pt-BR/docs/HTML/Introduction>)

Para iniciar uma tag HTML usa-se o < e para fechar />

```
<h1> Esse é um Titulo </h1>
<p> Esse é um paragrafo </p>
```

Esse é um título

```
<h1></h1>
```

Porém temos 6 níveis de títulos do h1 ao h6, sua numeração vai pela sua importância, no caso o h1 é o título mais importante da página.

A HTML consiste de um conjunto de elementos. Um elemento define o significado semântico do seu conteúdo. Elementos incluem tudo entre duas tags de elementos que casam entre si, incluindo as próprias tags. Por exemplo, o elemento "<p>" indica um parágrafo; o elemento "" indica uma imagem.

[Developer Mozilla Fundation](https://developer.mozilla.org/pt-BR/docs/HTML/Introduction) (<https://developer.mozilla.org/pt-BR/docs/HTML/Introduction>)

Você pode encontrar a lista de tags HTML nesse [link](https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element) (<https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element>)

Atributos

Uma Tag html pode possuir atributos, que são informações adicionais relacionadas aquela tag, atributos normalmente possuem duas partes:

- Nome do atributo
- Valor do atributo

```
<input type="text"/>
```

Nesse exemplo, a tag **input** possui o atributo **type** que identifica que o tipo do input é texto.

E para comentar no HTML usasse o seguinte código.

```
<!-- Esse é um comentário -->
```

Elementos Estruturais

O html sendo uma linguagem de marcação possui elementos que definem a estrutura de um documento, para isso usasse algumas tags para definir semanticamente o conteúdo.

Blocos de Informação

Algumas tags servem de "container" para se armazenar outras tags ou informações, a partir do HTML5 foi criado algumas tags para melhorar a semântica na Web

```
<div></div>

<!-- Tags adicionadas no HTML5 para dar mais semântica ao html -->
<main></main>
<section>
  <header></header>
  <article></article>
  <footer></footer>
</section>
<aside></aside>
<nav></nav>
```

[Exercício]

Para fixar o que aprendemos, imagine que você quer estruturar um código de uma postagem de blog, onde a postagem é uma sessão que contém um cabeçalho, onde ficará o título da postagem, após o cabeçalho teremos o artigo em si, e por fim o rodapé onde estará os dados como quem criou a postagem

Listas

Para identificar listas no HTML usasse a tag "ol" e "ul" onde "ol" é para referenciar listas **ordenadas** e "ul" para listas **não ordenadas**, para criar itens na lista usasse a tag "li".

Exemplo de listas

```
<!-- Ordenada -->
<ol>
  <li> Item 1 </li>
  <li> Item 2 </li>
  <li> Item 3 </li>
</ol>

<!-- Não Ordenada -->
<ul>
  <li> Item 1 </li>
  <li> Item 2 </li>
  <li> Item 3 </li>
</ul>
```

[Exercício]

Crie uma lista ordenada para o processo de troca de um pneu, e crie uma lista não ordenada para os itens de uma compra

Tabelas

Para definir uma tabela no HTML, usasse a tag "table" e para adicionar conteudo a essa tabela existem as tags "tr" e "td" e "th", porém como a Tabela é um bloco de informação ela possui tags de semântica para definir cabeçalho e rodapé da tabela por exemplo, para isso temos "thead" para definir o cabeçalho e "tfoot" para definir o rodapé.

Exemplo de Tabelas

```
<table>
  <!-- Cabeçalho da tabela -->
  <thead>
    <tr>
      <th>Nome</th>
      <th>Salario</th>
    </tr>
  </thead>
  <!-- Rodapé da tabela -->
  <!-- Mesmo colocando o tfoot a cima do tbody a tabela identifica que ele pertence ao rodapé
  da tabela e o adiciona somente no fim da mesma -->
  <tfoot>
    <tr>
      <td>Total dos pagamentos</td>
      <td>2200,00</td>
    </tr>
  </tfoot>
  <!-- Corpo da tabela -->
  <tbody>
    <tr>
      <td>João</td>
      <td>1000,00</td>
    </tr>
    <tr>
      <td>José</td>
      <td>1200,00</td>
    </tr>
  </tbody>
</table>
```

[Exercício]

Crie uma tabela de produtos e seus valores

Elementos Textuais

Assim como para definir bloco de informações, existem tags específicas para definir conteúdo "texto" dentro de um documento.

Segue alguns exemplos de elementos textuais

```
<h1>Titulo</h1>
<h2>Subtitulo</h2>
<h3>Titulo nível 3</h3>
<h4>Titulo nível 4</h4>
<h5>Titulo nível 5</h5>
<h6>Titulo nível 6</h6>

<p>Um paragrafo contendo texto em <b>Negrito</b> e em <i>italico</i> </p>
<blockquote> Uma citação de alguém importante </blockquote>
<span> No html entende-se o span</span> como um trecho muito curto de texto
```

```
<a href="http://"> link </a>
```

Formulário

Antes de explicar veja a estrutura a baixo

```
<form action="/contato.php" method="post">
  <input type="text" name="nome" placeholder="Digite um nome"/>
  <input type="email" name="email" placeholder="Digite seu e-mail"/>
  <textarea name="comentario"></textarea>
  <input type="reset" value="Limpar Formulário"/>
  <input type="submit" value="Submeter"/>
</form>
```

Agora vamos entender o que aconteceu, a tag **form** é uma tag de bloco de conteúdo e ela informa ao documento que ali dentro terá um formulário, criando assim um pequeno "escopo", ela tem dois atributos importantes o **action** que aponta para onde o formulário será enviado e o **method** que informa qual o método de submissão do formulário (o HTML por padrão aceita apenas GET e POST)

Para adicionar informações ao formulário usasse a tag **input**, para identificar esse input usasse o atributo **name** que como o nome diz, nomeia a tag para quando o formulário ser submetido o programador backend conseguir ler as informações de cada tag através do name da mesma, além do name o input possui algumas variações definidas dentro do atributo **type**, entre elas possuimos

```
<input type="text" name="" />
<input type="password" name=" " />
<input type="submit" name="" />
<input type="reset" name="" />
<input type="button" name="" />
<input type="radio" name="" />
<input type="checkbox" name="" />
<textarea></textarea>
<select></select>

<!-- Adicionadas ao HTML5 --&gt;
&lt;input type="range" name="" /&gt;
&lt;input type="number" name="" /&gt;
&lt;input type="date" name="" /&gt;
&lt;input type="email" name="" /&gt;</pre>
```

Cada tipo, tem sua finalidade, exemplo o tipo e-mail, faz uma pequena validação para que apenas seja aceito textos no formato de e-mail dentro daquele campo, ajudando nessa forma a facilitar na hora de validar os dados que são enviados através do Input.

Além da propriedade type e name o input possui mais alguns atributos interessantes, como **placeholder** (Adiciona uma marca d'água enquanto não possuir informação digitada), **required** (Identifica aquele campo do formulário como requerido), **disabled** (Desabilita um input para que não possa haver interação com o mesmo)

Para gerarmos ações no formulário existem dois inputs, o **reset** (para limpar o formulário) e **submit** (para submeter o formulário para o servidor).

[Exercício]

Para fixar o que aprendemos, vamos criar um arquivo HTML que possuirá um formulário de contato

```
<div style="page-break-after: always;"></div>
```

HTML Semântico

É só dos sentidos que procede toda a autenticidade, toda a boa consciência, toda a evidência da verdade.

"Nietzsche , Friedrich"

Semântica?

Semântica na vida real é um ramo da linguística que estuda significado das palavras, frases e textos de uma língua. É o estudo do significado.

Realidade em alguns casos

Vemos alguns desenvolvedores que preocupam-se com JavaScript, CSS e outros, portanto esquecem de uma premissa básica que é o significado correto das marcações em uma página.

Problemas

- Com a rápida expansão da internet um projeto que antes tinha o intuito de apenas compartilhar e organizar links na web criou proporções gigantes. Isso gerou grandes volumes conteúdos gerando um caos de informações. O volume é tão grande que "poluiu" os resultados e a navegação fica confusa e dispersa.
- Quando acessamos um site conseguimos distinguir qual é o cabeçalho do site? salvo a parte visual que nos ajuda identificar um logo com um menu que na maioria das vezes está sempre localizado na parte superior da página. Então sabemos que ali é um topo, mas os motores de busca e/ou leitores de tela não conseguem identificar isso.
- Antes utilizávamos apenas `<div>` para estruturar e demarcar nossas páginas, mas esse elemento não tem nenhuma semântica, usando `<div id="header">`, podemos acessar este elemento, portanto os motores de busca e leitores de tela não conseguem enxergar isso como um cabeçalho é apenas uma div.

Solução

- Web semântica é um projeto com objetivo de aplicar conceitos inteligentes na internet. Nela cada informação vem com um significado bem definido permitindo melhor interação com o usuário, novos motores de busca(que marcam a relevância em um a página), interfaces inovadoras, ou seja uma organização inteligente de conteúdos.
- Devemos fazer o uso do HTML5 que nos fornece tags semânticas, para demarcar um página web.

Porque usar HTML5 semântico e sua importância

- Torna mais fácil os leitores de tela interpretarem informação de um site, usando html semântico podemos marcar as páginas e torna-las mais acessíveis tanto para pessoas com deficiência quanto para os motores de busca.
- Transmitimos significados através das tags em uma página.
- É graças a semântica que sua página vai indexar legal ou não nos buscadores. Não somente por isso, a semântica ajuda na acessibilidade de sua página ou aplicação

Exemplo na Vida Real e no HTML5

- Na real, quando você ler um livro em voz alta e depara-se com uma (,), você sabe que alí uma parte da frase.
- Logo em seguida rapidamente retomamos a leitura mas nesse tempo temos uma pequena pausa.
- Quando encontramos um ponto final, a pausa é maior e note que isso nos guia em nossa leitura.
- Imagine em uma página de um site com diversas áreas, usamos as tags semânticas para demarcar partes da sua página os motores de buscas vão saber identificar qual é o cabeçalho da página, qual é o menu principal, qual é a parte principal do seu site, ou se o seu site possui várias seções, qual é o rodapé da página e etc...

Porque estruturar?

```
<h1> Title </h1>
```

```
<p> Lorem ipsum...</p>
```

- Sabemos que o H1 sempre é o título mais importante da página mas o local onde ele está isso que faz a diferença, ou seja essa estrutura facilita na os motores e leitores de tela identificarem do que se trata cada conteúdo na página.

Novos Elementos Estruturais HTML5

- Algumas Novas tags do HTML5 que nos ajudam a organizar e estruturar o conteúdo de nossos sites. Estas novas tags vieram para melhorar a semântica dos elementos estruturais do código

```
<header>
```

```
<header>
  <h1><a href="/">Company Name</a></h1>
  <nav>
    <ul>
      <li><a href="index.html">Home</a></li>
      <li><a href="/about/">About</a></li>
      <li><a href="/blog/">Blog</a></li>
    </ul>
  </nav>
</header>
```

```
<nav>
```

Define um grupo ou bloco de links de navegação.

```
<nav>
  <ul>
    <li><a href="index.html">Home</a></li>
    <li><a href="/about/">About</a></li>
    <li><a href="/blog/">Blog</a></li>
  </ul>
</nav>
```

```
<footer>
```

Define o rodapé das seções ou da página.

```
<footer>
  <ul>
    <li>copyright</li>
    <li>sitemap</li>
    <li>contact</li>
    <li>to top</li>
  </ul>
</footer>
```

```
<aside>
```

- Define um elemento lateral que pode conter blocos de navegação (NAVs), citações e outras informações que costumamos colocar em uma sidebar.
- Servem para chamar sua atenção para alguma informação importante ou outras informações que agregarão mais ao conteúdo principal.
- Pode estar relacionado a um post, a página completa, etc.

Neste exemplo temos um aside relacionado a uma artigo.

```
<!-- Aside -->
<aside>
  <h3>Foodblogs I like</h3>
```

```

<a href="http://www.bakerella.com">Bakerella</a>
<a href="http://sourdough.com/"> Sourdough.com</a>
<a href="http://www.bakingobsession.com">BakingObsession</a>
</aside>
<!-- Aside -->
<article>
  <header>
    <h1>All About Flour</h1>
    <p class="byline">by Jane Doe</p>
  </header>
  <section>
    <h2>The Two Types of Wheat</h2>
    <p>There ... to rise.</p>
    <p>Where ... with less protein.</p>
  </section>
</article>

```

<article>

Define a área onde há um artigo, texto, redação, conteúdo e etc...

```

<article>
  <h1>Apple</h1>
  <p>The <b>apple</b> is the pomaceous fruit of the apple tree...</p>
  ...
</article>

```

<section>

- Define um bloco ou um grupo de um assunto específico.
- É importante entender que a section agrupa diversos elementos que tenham relação entre si. Por exemplo, se há uma área no site que há links, conteúdo, imagens e etc de um assunto em comum, você agrupará esses elementos com uma section. Nesse caso, ele entrou no lugar daquele div que fazíamos para dividir grandes blocos de assuntos em comum.

```

<section>
  <h2> Title section </h2>
  Section content appears here.
</section>

```

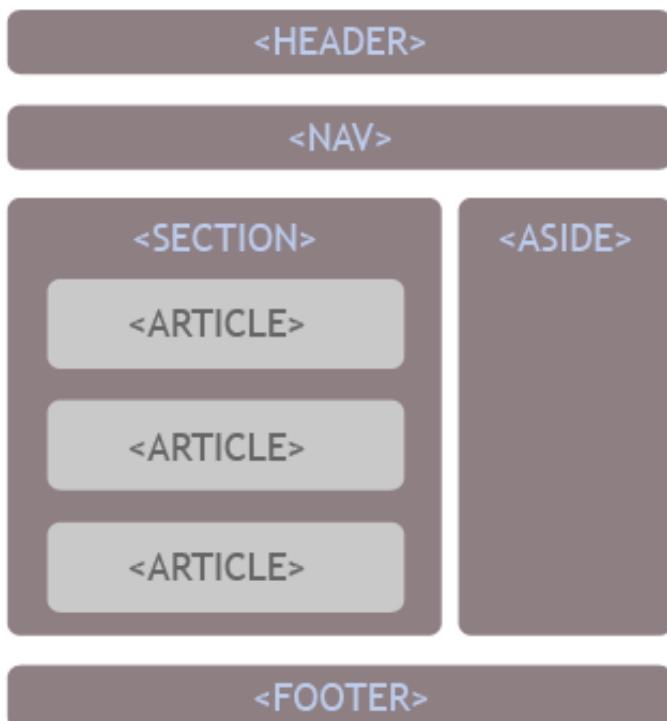
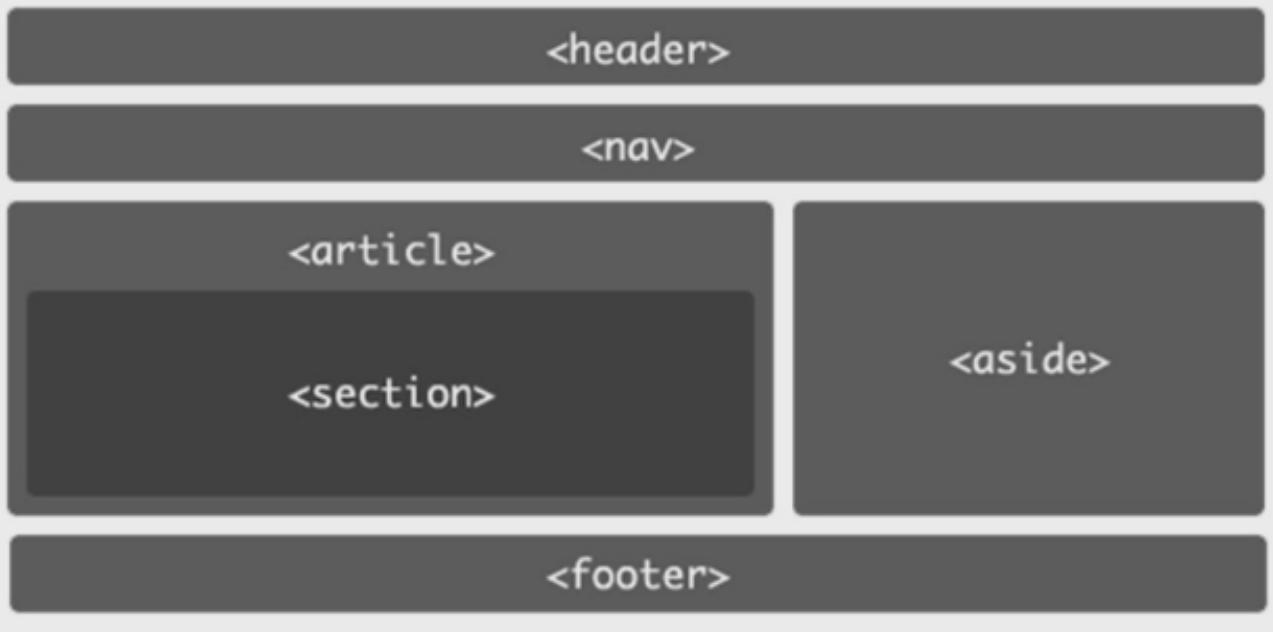
Outros elementos estruturais HTML5

Tags	Descrição
<mark>	Destacar um texto
<progress>	Progresso de uma tarefa
<hgroup>	Usado em uma seção de títulos, usando <h1> para <h6>, onde o maior é o título principal da seção, e os outros são sub-títulos
<details>	Descreve detalhes de um documento ou parte
<summary>	Um caption ou sumário, dentro da tag <details>

Exemplos de Simples Estruturas



Semantics via HTML5



[Exercício]

Para concluir, vamos construir o HTML referente a essa imagem

```
<div style="page-break-after: always;"></div>
```

Introdução à CSS

Seletores CSS básicos

Seletores estão entre as primeiras coisas que você aprende quando começa a estudar CSS. Sem dúvida os seletores fazem parte dos assuntos fundamentais das CSS, contudo poucos desenvolvedores sabem tirar proveito de todo o seu potencial. Ainda que você possa fazer muitas estilizações com os seletores do tipo ID e os seletores de classes, há muito mais a fazer com seletores.

Seletores CSS permitem que você selecione e manipular elementos HTML. Seletores CSS são usados para "encontrar" (ou selecione) elementos HTML com base em sua ID, classe, tipo, atributo, Seletores CSS permitem que você selecione e manipular elementos HTML. Seletores CSS são usados para "encontrar" (ou selecione) elementos HTML com base em sua ID, classe, tipo, atributo, e muito mais. e muito mais.

Vamos aos fundamentos básicos. Um seletor CSS é uma declaração em um formato que "casa" com todos os elementos que sigam aquele formato na árvore do documento. Quando todas as condições estabelecidas no formato da declaração são satisfeitas o seletor "casa" com o elemento (ou elementos) no documento e as regras escritas no seletor são aplicadas. Considere a regra CSS bem simples escrita a seguir:

```
p { color:#f00; }
```

O seletor é a parte da regra CSS que está antes do sinal “{“ (chave de abertura). O seletor aqui é p, que "casa" com todos os elementos p do documento e faz com que qualquer texto dentro de um parágrafo seja na cor vermelha. Bem básico.

Seletores Visão geral

Explicarei detalhadamente cada um destes seletores nas duas primeiras partes deste artigo, assim, continue lendo. Alguns termos usados na tabela acima e ao longo do artigo necessitam de uma explicação adicional:

Descendente

Um elemento que é filho, neto ou descendente mais distante de um elemento, na árvore do documento.

Ancestral

Um elemento que é pai, avô ou ancestral mais distante de um elemento na árvore do documento.

Filho

O descendente direto de um elemento. Nenhum elemento existe entre os dois na árvore do documento.

Pai

O ancestral direto de um elemento. Nenhum elemento existe entre os dois na árvore do documento.

Sibling (irmãos)

Elementos irmãos, filhos do mesmo pai.

Seletores Simples e combinados

Existem duas categorias básicas de seletores: os simples e os combinados.

Um seletor simples consiste em um tipo qualquer de seletor ou o seletor universal seguido por nenhum ou algum seletor de atributo, seletor tipo ID, seletor de classe ou pseudo-classe. A seguir uma regra contendo um exemplo de seletor simples:

```
p.info { background:#ff0; }
```

Um seletor combinado (algumas vezes chamado de seletor contextual) consiste de dois ou mais seletores simples separados por um elemento de combinação. A seguir um exemplo de seletor combinado.

```
div p { font-weight:bold; }
```

A regra acima aplica-se a todo elemento p que seja descendente do elemento div.

Um pseudo-elemento pode ser colocado como apêndice a um seletor. Em seletores combinados, o pseudo-elemento somente poderá ser adicionado como apêndice ao último seletor simples.

Mais a frente serão detalhados com mais profundidade os seletores combinados, os elementos de combinação e os pseudo-elementos.

Seletores Universal

O seletor universal é representado por um asterisco, “*”, e casa com todos os elementos do documento. É raro ver-se empregado em uma folha de estilos, mas o seletor universal é muito usado com seletores tipo ID e seletores de classe. Se o seletor universal não for o único componente de um seletor simples, o “*” não deve ser usado :

- .myclass é equivalente a *.myclass
- #myid é equivalente a *#myid

Um uso bastante popular para o seletor universal é o uso para zerar margens e paddings de todos os elementos do documento:

```
* { margin:0; padding:0; }
```

Este procedimento é também conhecido como Global White Space Reset.

Seletores Tipo

Um seletor tipo, casa com qualquer instância de um determinado tipo de elemento. A regra a seguir casa com qualquer elemento (do tipo) parágrafo no documento e configura seu tamanho de fonte para 2em:

```
p { font-size:2em; }
```

Seletor – classe

O seletor de classe é representado por um ponto, “.”, e tem como alvo elementos com um determinado valor para seu atributo class. A regra a seguir aplica-se a todo elemento parágrafo cuja classe tenha o nome “info”:

```
p.info { background:#ff0; }
```

Você pode atribuir vários nomes para a classe de um elemento – o atributo class pode conter uma lista de vários nomes separados por espaço em branco. Assim, os seletores de classe podem ser usados para casar com elementos cuja classe contenha vários nomes. A regra a seguir casa com elementos p que tenham os nomes “info” e “error” declarados em seu atributo class:

```
p.info.error { color:#900; font-weight:bold; }
```

O tipo de elemento não precisa necessariamente ser declarado. Este procedimento, não declarar o tipo de elemento, equivale a usar o seletor universal como tipo de elemento. A regra a seguir casa com qualquer elemento da classe “info”, independentemente do tipo de elemento:

```
.info { background:#ff0; }
```

Seletor – ID

O seletor ID é representado por um sinal de “trilha” (ou “jogo da velha”), “#”, e tem como alvo elementos com um determinado valor de atributo ID. A regra a seguir aplica-se a todos os elementos cujo nome de ID seja “info”, independentemente do tipo de elemento:

```
#info { background:#ff0; }
```

Se você especificar um determinado tipo de elemento a regra será aplicada somente àquele tipo de elemento que tenha o nome da ID especificado:

```
p#info { background:#ff0; }
```

É importante lembrar que seletores ID tem uma especificidade maior que seletores de classe e que um valor de ID deve ser único em um mesmo documento. Assim um determinado seletor ID será aplicável a um único elemento no documento.

Elementos de combinação

Elementos de combinação de seletores são usados para separar dois ou mais seletores simples que compõem um seletor combinado. Os elementos de combinação disponíveis são: espaço em branco (qualquer quantidade de espaço, tabulação ou caracteres de espaçamento), o sinal de maior “>” e o sinal de adição “+”. A função de cada um destes elementos de combinação dos seletores será descrita adiante.

Seletores descendentes

Um seletor descendente é uma combinação de dois ou mais seletores simples separados por um espaço em branco. Caso com elementos que sejam descendentes do primeiro elemento simples declarado no seletor. Por exemplo, na regra a seguir o seletor casa com todos os elementos que sejam descendentes do elemento div:

```
div p { color:#f00; }
```

Cada um dos seletores que compõem um seletor descendente pode ser um seletor simples de qualquer natureza. Na regra a seguir o seletor casa com todo o elemento p da classe info contido em um elemento li que esteja contido em um elemento div cuja id seja myid.

```
div#myid li p.info { color:#f00; }
```

Seletores descendentes permitem que você case um elemento sem necessidade de atribuir-lhe uma classe ou uma id, o que resultará em uma marcação mais limpa. Vamos supor uma lista de navegação conforme a marcação abaixo:

```
<ul id="nav">
  <li><a href="#">Link 1</a></li>
</ul>
<ul>
  <li><a href="#">Link 2</a></li>
  <li><a href="#">Link 3</a></li>
</ul>
```

Para atingir os itens de lista e links contidos na lista de navegação você poderia usar as seguintes regras CSS:

```
#nav li { display:inline; }
#nav a { font-weight:bold; }
```

Estas regras não serão aplicadas a nenhum outro item de lista ou links dentro do documento. Agora compare com a opção de nomear uma classe para cada item da lista e para os links e você perceberá quão mais limpa poderá tornar-se sua marcação com o uso de seletores descendentes

Seletores Filho

Um seletor filho tem como alvo um filho imediato de um elemento. O seletor filho consiste de um ou mais seletores simples separados por um sinal de maior “>”. O elemento pai fica à esquerda do sinal “>”, e é permitido deixar espaço em branco entre o elemento de combinação e os seletores. A regra a seguir aplica-se a todos os elementos strong que sejam filhos de um elemento div:

```
div > strong { color:#f00; }
```

Somente elementos strong que sejam descendentes diretos do elemento div serão afetados por esta regra. Se houver qualquer outro elemento entre o elemento div e o elemento strong na árvore do documento, o seletor não se aplicará. No exemplo a seguir, somente “Texto um” será afetado pela regra:

```
<div>
  <strong>Texto um</strong>
  <p><strong>Texto dois</strong></p>
</div>
```

Seletores Irmãos adjacentes (sibling selectors)

Um seletor filho tem como alvo um filho imediato de um elemento. O seletor filho consiste de um ou mais seletores simples separados por um sinal de maior “+”. O elemento pai fica à esquerda do sinal “+”, e é permitido deixar espaço em branco entre o elemento de combinação e os seletores. A regra a seguir aplica-se a todos os elementos strong que sejam filhos de um elemento div:

```
p + strong { color:#f00; }
```

Somente elementos strong que sejam irmãos diretos do elemento p serão afetados por esta regra. Se houver qualquer outro elemento entre o elemento + e o elemento strong na árvore do documento, o seletor não se aplicará. No exemplo a seguir, somente “Texto dois” será afetado pela regra:

```
<div>
  <p><strong>Texto um</strong></p>
  <strong>Texto dois</strong>
</div>
```

Agrupando seletores

Eu decidi abordar o agrupamento a esta altura do artigo, porque um erro comum que eu vejo as pessoas cometer quando estão aprendendo CSS diz respeito ao agrupamento de seletores.

Para aplicar uma mesma regra a diferentes elementos alvo casados por diferentes seletores você pode agrupar os seletores em uma lista e separando-os por uma vírgula no lugar de escrever repetidamente a mesma regra para cada um dos seletores. O erro que muitos cometem é o de não listar de modo completo todos os seletores. Considere a seguinte marcação:

```
<div id="news">
  <h3>News</h3>
  <ul>
    <li>Item 1</li>
    <li>Item 2</li>
  </ul>
</div>
```

Agora considere que você quer aplicar a mesma margem para cabeçalhos do nível 3 e para listas não ordenadas que estejam dentro do elemento div cuja id é “news”. Aqui maneira errada:

```
div#news h3, ul { margin:0 2em; }
```

Esta regra será aplicada a ambos os elementos h3 e ul na div#news. O problema é que atingirá todos os elementos ul contidos no documento, e não apenas aqueles na div#news.

Agora a maneira correta de agrupar os seletores para este caso:

```
div#news h3, div#news ul { margin:0 2em; }
```

Assim, quando agrupar seletores lembre-se de escrever por completo cada um deles.

Seletores de Atributo

Seletores de atributo atingem elementos baseados no valor de atributo declarado no seletor. Existem quatro maneiras de declarar um seletor de atributo:

- [att] Casa com qualquer elemento com o atributo att independente do seu valor.
- [att=val] - Casa com qualquer elemento com o atributo att cujo valor seja “val”.
- [att~=val] - Casa com qualquer elemento que tenha um atributo att de valor igual a um valor qualquer separado por um espaço de um valor igual “val”. Neste caso “val” não pode conter espaços.
- [att^=val] - Casa com qualquer elemento que tenha um atributo att de valor igual a um valor qualquer separado por um hífen de um valor começando com “val”. O principal uso deste seletor é o de casar elementos com um valor de idioma especificado no atributo lang (xml:lang em XHTML), por exemplo;“en”, “en-us”, “en-gb”, etc.

Exemplos

O seletor na regra a seguir casa com todos os elementos p que tenham o atributo title, independentemente do valor do atributo:

```
p[title] { color:#f00; }
```

No próximo exemplo o seletor casa com todos os elementos div que tem um valor para o atributo class igual a error:

```
div[class=error] { color:#f00; }
```

Para atingir todos os elementos td cujo atributo headers contenha o valor “col1”, podemos usar o seguinte seletor:

```
td[headers~=col1] { color:#f00; }
```

E finalmente, o seletor seguinte atinge todo elemento p cujo atributo lang comece com en:

```
p[lang|=en] { color:#f00; }
```

Múltiplos seletores de atributos podem ser usados em um mesmo seletor. Isto possibilita atingir vários diferentes atributos para o mesmo elemento. A regra a seguir aplica-se a todos os elementos blockquote que tenham o atributo class de valor igual a "quote", e mais o atributo cite (independentemente do seu valor):

```
blockquote[class=quote][cite] { color:#f00; }
```

Seletores Pseudo-elementos

Como o conteúdo iria se extender demais se abordassemos mais esse tópico, por isso quero deixar avisado aqui que a continuação dessa parte do CSS será melhor explicada em materiais futuros que você poderá estudar por aqui.

```
<div style="page-break-after: always;"></div>
```

Introdução ao Javascript

História do JavaScript

JavaScript foi originalmente desenvolvido por Brendan Eich da Netscape sob o nome de Mocha, posteriormente teve seu nome mudado para LiveScript e por fim JavaScript. LiveScript foi o nome oficial da linguagem quando foi lançada pela primeira vez na versão beta do navegador Netscape 2.0 em setembro de 1995, mas teve seu nome mudado em um anúncio conjunto com a Sun Microsystems em dezembro de 1995 quando foi implementado no navegador Netscape versão 2.0B3. A mudança de nome de LiveScript para JavaScript coincidiu com a época em que a Netscape adicionou suporte à tecnologia Java em seu navegador (Applets). A escolha final do nome causou confusão dando a impressão de que a linguagem foi baseada em java, sendo que tal escolha foi caracterizada por muitos como uma estratégia de marketing da Netscape para aproveitar a popularidade do recém-lançado Java. JavaScript rapidamente adquiriu ampla aceitação como linguagem de script client-side de páginas web. Como consequência, a Microsoft desenvolveu um dialeto compatível com o próprio JavaScript, mas que levou o nome de JScript para evitar problemas de trademark. JScript foi incluído no Internet Explorer 3.0, liberado em Agosto de 1996. Em novembro de 1996 a Netscape anunciou que tinha submetido o JavaScript para Ecma internacional como candidato a padrão industrial e o trabalho subsequente resultou na versão padronizada chamada ECMAScript. O JavaScript tem se transformado na linguagem de programação mais popular da web. Inicialmente muitos profissionais denegriram a linguagem, pois a mesma tinha como alvo principal o público leigo. Com o advento do Ajax, o JavaScript teve sua popularidade de volta e recebeu mais atenção profissional. O resultado foi a proliferação de frameworks e bibliotecas, práticas de programação melhoradas e o aumento no uso do JavaScript fora do ambiente de navegadores bem como o uso de plataformas de JavaScript server-side.

Onde podemos usar?

Inicialmente o Javascript era utilizado apenas nos navegadores, porém hoje em dia com a evolução das engines de Javascript como SpiderMonkey e V8, eles levaram o Javascript também para o lado do servidor com o Node.js e bancos NoSQL que utilizam Javascript como CouchDb e MongoDb.

O que oferece?

O JavaScript além de ser a linguagem mais usada no Universo nos oferece algumas coisas interessantes que sem elas a Internet como existe hoje não seria possível:

- Dinamismo
- Validação de Formulários
- Interatividade
- Controle de Comportamento
- Personalização da página

Atualmente o JavaScript é o motor da Internet, principalmente com o advento do AJAX que fez nossas interfaces ficarem mais ricas e interativas. Caso você não saiba o Facebook só existe do jeito como é graças a ele.

Principais Características

- Imperativa;

- Funcional;
- Estruturada;
- Fracamente Tipada;
- Tipagem dinâmica;
- Orientada à Objetos;
- Baseada em Protótipos;
- Case Sensitive.

JavaScript suporta elementos de sintaxe de programação estruturada da linguagem C (por exemplo, if, while, switch). Uma exceção é a questão do escopo: o escopo em blocos ao estilo do C não é suportado, em seu lugar o JavaScript utiliza as funções como delimitadores de escopo. Assim como C, JavaScript faz distinção entre expressões e comandos(statement). Uma diferença sintática do C é que a quebra de linha termina automaticamente o comando, sendo muitas vezes o ponto-e-vírgula opcional ao fim do comando. Funções aninhadas Funções 'internas' ou 'aninhadas' são funções definidas dentro de outras funções. São criadas cada vez que a função que as contém (externa) é chamada. Além disso, o escopo da função externa, incluindo constantes, variáveis locais e valores de argumento, se transforma parte do estado interno de cada objeto criado a partir da função interna, mesmo depois que a execução da função interna é concluída. Dinâmica

Tipagem dinâmica

Como na maioria das linguagens de script, tipos são associados com valores, não com variáveis. Por exemplo, uma variável x poderia ser associada a um número e mais tarde associada a uma string. Isso permite que o JavaScript suporte várias formas de testar o tipo de um objeto, incluindo o duck typing. JavaScript inclui a função eval que consegue executar em tempo de execução expressões e comandos da linguagem que estejam escritos na string passada como argumento.

Funcional

Funções de primeira classe No JavaScript as funções são de primeira classe, isto é, são objetos que possuem propriedades e métodos, podem ser passados como argumentos, podem ter suas referências armazenadas em variáveis e retornados como qualquer outro objeto. Funções anônimas São funções, como o próprio nome já diz, que não possuem nome, são normalmente criadas apenas para uma finalidade e muito utilizadas em callbacks.

Orientada à objetos

JavaScript é quase inteiramente baseada em objetos. Objetos JavaScript são arrays associativos, que também respondem aos mapeamentos de seus protótipos, algo similar à ideia de herança entre classes, porém baseada em protótipos. Os nomes das propriedades de um objeto são strings, o que permite o acesso por duas possibilidades de sintaxe. Por exemplo: obj.x = 10 e obj["x"] = 10 são equivalentes, o ponto neste exemplo é apenas açúcar sintático. Propriedades e seus valores podem ser adicionadas, mudadas, ou deletadas em tempo de execução. A maioria das propriedades de um objeto (e aqueles em sua cadeia de herança via protótipo) pode ser enumerada usando-se uma estrutura de repetição for...in. Javascript possui um pequeno número de objetos padrão da linguagem como window e document.

Baseada em Protótipos

JavaScript usa protótipos em vez de classes para o mecanismo herança. É possível simular muitas características de orientação a objetos baseada em classes com protótipos. Funções e métodos Diferente de muitas linguagens orientadas a objetos, não há distinção entre a definição de uma função e a definição de um método no JavaScript. A distinção ocorre durante a chamada da função; a função pode ser chamada como um método. Quando uma função é chamada como método de um objeto, a keyword this da função é associada àquele objeto via tal invocação.

Detalhes

O JavaScript é case sensitive, ou seja, tem diferenciação entre maiúsculas e minúsculas, portanto cuidado ao nomear suas variáveis e funções.

O uso do ponto e vírgula (;) no final do comando é facultativo, porém as boas práticas nos dizem para **SEMPRE** usar, ok?!

Para fazermos um comentário de linha basta adicionar // no inicio do comentário. Exemplo:

```
var nome = "Suissa"; //meu nome
```

E para fazer um comentário de bloco deve-se iniciar com /* e finalizar com */. Exemplo:

```
/*
```

```
@author: Suissa  
@curso: JS4Girls  
*/
```

Javascript não é Java



Por que um tópico para explicar isso?

Porque a maioria das pessoas que não programa sempre faz confusão, porém as 2, apesar do nome, são quase o oposto. É como dizem:

Java está para o Javascript assim como Bola está para Bolacha.

Mas qual são as principais diferenças entre eles?

- Java é uma linguagem compilada, ao passo que JavaScript é uma linguagem interpretada;
- Java é fortemente tipado, enquanto que o JavaScript é fracamente tipado.

Isso só para citar as maiores diferenças, sem contar a sintaxe.

Exemplo Hello JS4Girls

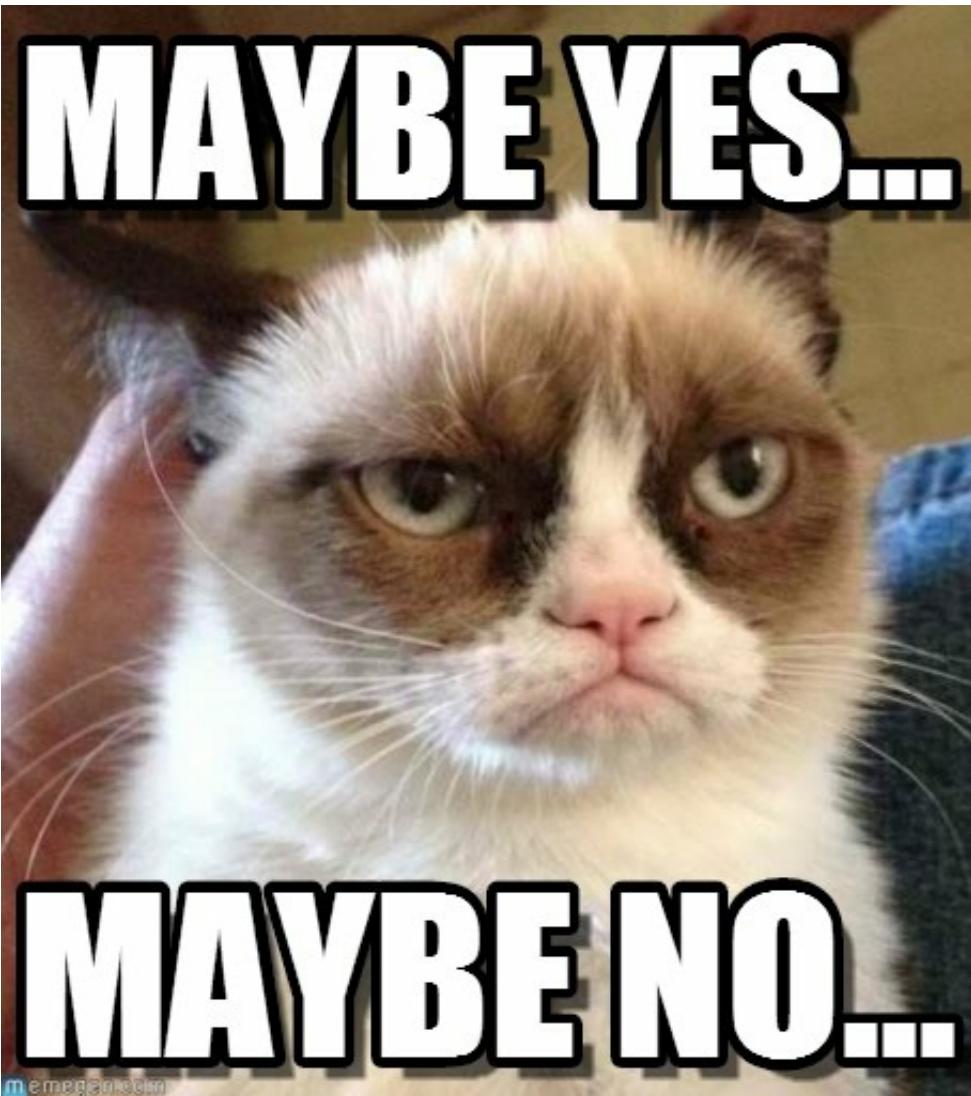
Agora abra o console do seu navegador, (pedir ajuda ao colega ou professor), e digite o seguinte comando:

```
alert("Hello JS4Girls!");
```

E execute apertando *ENTER*.

Console do Navegador

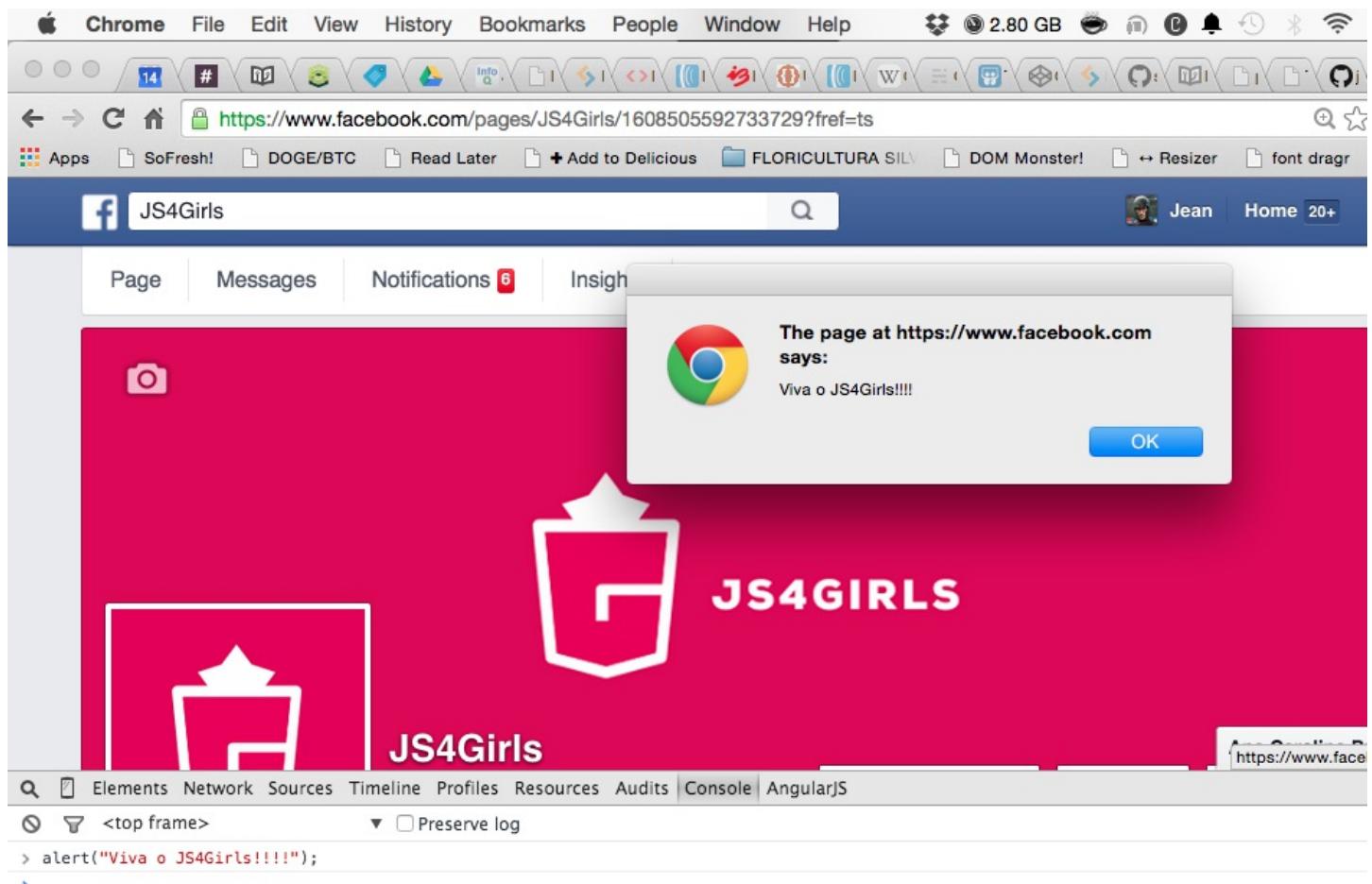
Provavelmente deve ser a primeira vez que você tem contato com esse tal de console correto?



Certo.

Bom o console, que existe tamb'pem em todos os outros navegadores importantes do mercado, é a interface onde podemos rodar comandos de JavaScript diretamente do seu navegador.

Você deve ter percebido que o comando `alert` faz abrir uma caixinha de alerta na minha tela correto?



Sim eu posso executar comandos diretamente em algum site aberto podendo modificar dinamicamente qualquer parte do mesmo, porém essa modificação só será vista por você na sua máquina e quando recarregar a página aquele seu código terá ido embora. Logo não é uma ferramenta para você zoar com outros sites, mas sim conseguir trabalhar de uma forma mais eficaz com o JavaScript.

Agora vamos conhecer mais uma função bem simples do JavaScript que faz abrir uma caixa de diálogo com um campo para entrada de algum valor.

Vamos rodar o seguinte código no nosso console:

```
prompt("Qual é sua idade?");
```

The page at <https://www.facebook.com> says:
Qual é sua idade?
30

Cancel OK

JS4GIRLS

Ana Carolina

Elements Network Sources Timeline Profiles Resources Audits Console AngularJS

<top frame> ▾ □ Preserve log

```
> prompt("Qual é sua idade?");  
>
```

Você deve se perguntar:

- Mas para onde vai esse valor da idade?

Ótima pergunta minha cara aluna, nesse código que rodamos ele não vai para lugar algum, então vamos corrigir isso definindo que o valor desse prompt vá para minha variável `idade` e depois eu mostre esse valor com `alert`.

```
var idade = prompt("Qual é sua idade?");  
alert("Minha idade é " + idade + " anos.");
```

Bem simples não? Agora de mãos desse conhecimento vamos resolver o exercício abaixo.

[Exercício] Escreva um código em que você receba seu nome e depois escreva com `alert` a mensagem: "Meu nome é " + nome

Introdução à Estrutura de Dados

O que é estrutura de dados?



Primeiramente precisamos saber o que é um dado, um dado nada mais é que um valor básico. Como assim?

Pense em um número?

Pensou?

Tá pode esquecer ele agora.

**PENSE EM 01 NÚMERO.
ADICIONE 42. SUBTRAIA 7.
ADICIONE 1.**

**FECHE OS OLHOS. É
ESCURO, NÉ?**

GERADORMEMES.COM

#brinks

Nesse caso o número é um dado do tipo Number ou inteiro, dependendo da linguagem. Levando isso em consideração então podemos deduzir que palavras também são dados, correto?

Corretíssimo! E esse tipo de dado se chama String, prazer.



quickmeme.com

Especificamente no JavaScript temos os seguintes tipos de dados:

- null;
- String;
- Number;
- Boolean;
- Array.

Com um adendo especial para o:

- undefined.

null / nulo

O valor `null` é um literal em JavaScript que representa um valor nulo ou "vazio" (p/ex: que aponta para um objeto inexistente).

undefined

O valor `undefined` representa um valor indefinido.

Diferenças entre null e undefined

Basicamente a diferença entre eles é que o `null` é um tipo de objeto e o `undefined` é um valor, podemos verificar isso analisando o código abaixo:

```
typeof null      // object
typeof undefined // undefined
null === undefined // falso
null == undefined // verdadeiro
```

String

String é o tipo utilizado para armazenar textos. Uma das operações mais usadas nas strings é checar seu tamanho, para concatená-las usamos os operadores + e +=. Para checar pela existência ou posição de substrings usamos o método indexOf e para extrair substrings com o método substring.

Para criarmos uma String podemos simplesmente atribuir o valor diretamente na variável.

```
var palavra = "JS4Girls";
typeof palavra; // "string"
```

Agora criando ele com o construtor String.

```
var palavra = new String("JS4Girls");
typeof palavra; // "object"
```

Acesso à um caractere

Há duas maneiras de acessar um caráter individual em uma string. A primeira é o método charAt:

```
return 'cat'.charAt(1); // retorna "a"
```

A outra maneira (introduzido no ECMAScript 5) consiste em tratar a string como um objeto Array-like, onde os caracteres individuais correspondem a um índice numérico:

```
return 'cat'[1]; // retorna "a"
```

Comparando strings

No JavaScript, basta usar o operador maior que e menor que:

```
var a = "a";
var b = "b";
if (a < b) // true
  console.log(a + " é menor que " + b);
else if (a > b)
  console.log(a + " é maior que " + b);
else
  console.log(a + " e " + b + " são iguais.");
```

[Exercício] PENSAR EM UM EXERCÍCIO SEM IF

Number

O tipo Number é utilizado, como vimos anteriormente, para armazenar números e para isso também temos duas formas diferentes de fazê-lo:

```
var numero = 420;
typeof numero; // "number"
```

Agora criando ele com o construtor Number.

```
var numero = new Number(420);
typeof numero; // "object"
```

Comparando números

Simples, só usar o operador maior que e menor que:

```
var a = 420;
var b = 666;
if (a < b) // true
  console.log(a + " é menor que " + b);
else if (a > b)
```

```
    console.log(a + " é maior que " + b);
else
    console.log(a + " e " + b + " são iguais.");
```

[Exercício] Escreva um código que receberá o ano de nascimento via `prompt` e teste se é o usuário é maior de idade, caso sim mostre a mensagem: "Pode entrar". Caso não, mostre: "Entrada NEGADA!"

Boolean

Ainda veremos no módulo de lógico o que um valor booleano significa, por hora basta saber que ele representa apenas dois valores possíveis:

- verdadeiro
- falso

E ele é a base da lógica booleana que aprenderemos a seguir. Para criarmos um valor desse tipo é bem simples.

```
var bool = false;
typeof bool; // "boolean"
```

Criando ele com o construtor Boolean.

```
var bool = new Boolean(false);
typeof bool; // "object"
```

O valor passado como primeiro parâmetro é convertido para um valor booleano, se necessário. Se o valor é omitido ou é 0, -0, null, false, NaN, undefined ou é uma string vazia(""), o objeto terá um valor inicial de false. Todos outros valores, incluindo qualquer objeto ou string "false", criam um objeto com valor inicial true.

Não confunda os valores primitivos Boolean true e false com os valores true and false do objeto Boolean.

Qualquer objeto cujo o valor não é undefined ou null, incluindo um objeto Boolean que o valor seja false, é avaliado para true quando passa por uma declaração condicional. Por exemplo, a condição a seguir if a declaração é avaliada como true:

```
var x = new Boolean(false);
if (x) {
    // this code is executed
}
```

Esse comportamento não se aplica aos primitivos Boolean. Por exemplo, a condição a seguir if a declaração é avaliada como false:

```
var x = false;
if (x) {
    // this code is not executed
}
```

Caso você queira converter algum valor para booleano prefira fazer sem o contrutor `new`, como no código abaixo:

```
var x = Boolean(expression);      // preferido
var x = new Boolean(expression); // não use

// Exemplo

var bool = Boolean("JS4Girls");
console.log(bool); // true

var bool = Boolean(0);
console.log(bool); // false
```

[Exercício] Escreva um código que receba uma variável numérica e teste esse valor se é verdadeiro, utilizando as 2 formas acima.

Array / Matriz

Array, ah esse Array!

Nunca esquecerei uma aula que tive no segundo ano do ensino médio e a aula de matemática era sobre **Arrays**.

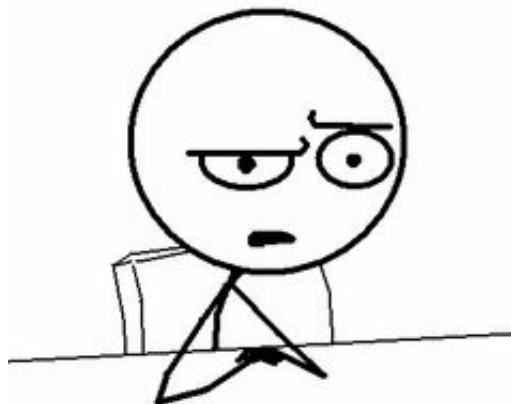
□

Lembro ainda como se fosse hoje, perguntei ao professor:

- Professor mas onde que eu vou usar isso na vida?

Típica pergunta de aluno chato né? Então, olha a resposta do infeliz:

- Não sei.

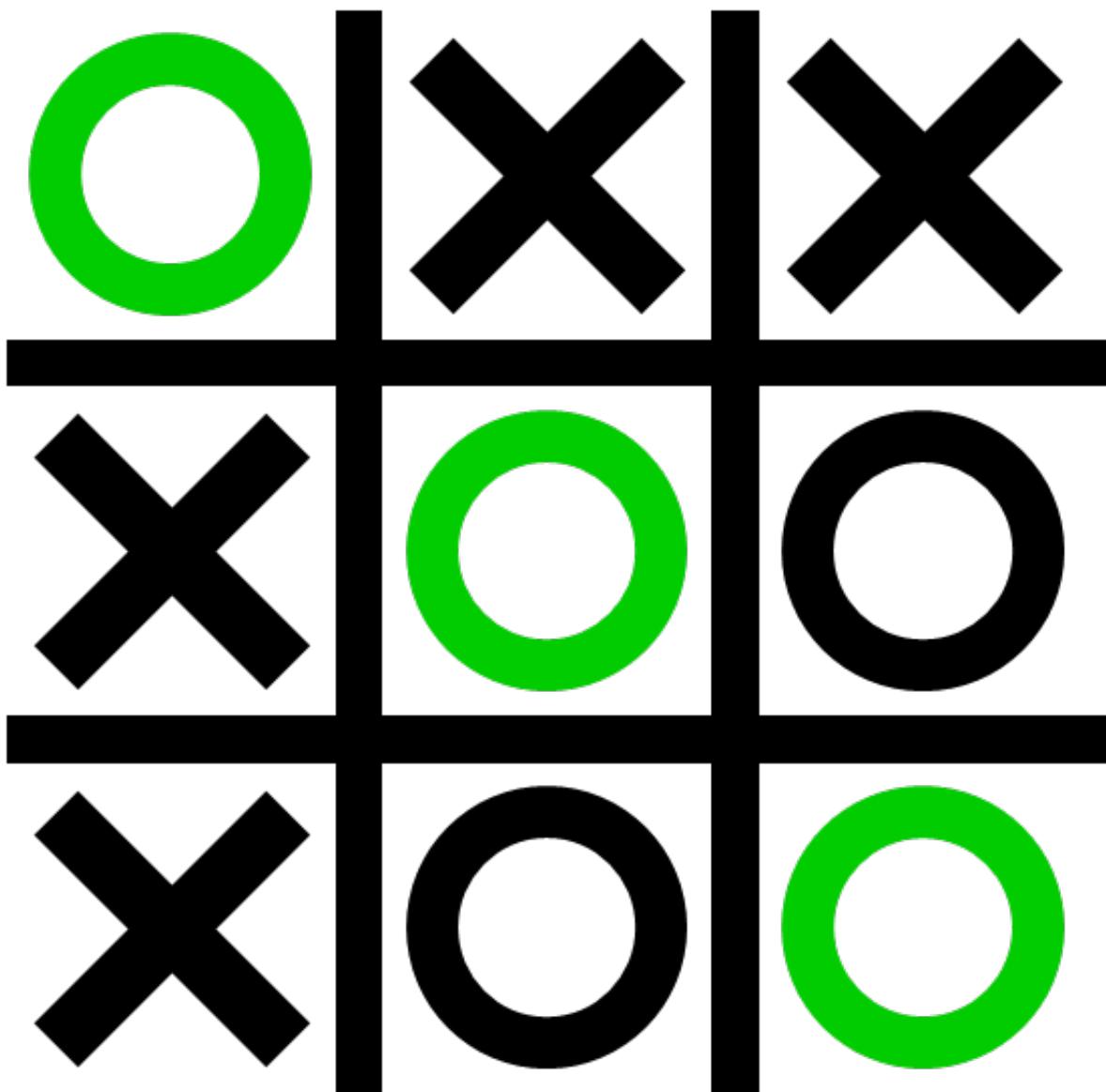


Pois é, agora imagine se ele tivesse me perguntado o que eu iria fazer da minha vida, já que naquela época eu já estava começando na programação, com 14 anos.

Agora sabendo disso espero que você dê muita atenção aos *Arrays* pois ele são a base das estruturas de dados utilizadas na programação, é praticamente impossível você programar sem utilizar algum tipo de *Array*.

Beleza, mas e o que são esses assustadores *Arrays*?

Imagine o Jogo da Velha.



Agora imagine que cada quadradinho onde podemos colocar um X ou O é uma posição nesse *Array*, que nesse caso é uma matriz quadrada, já que tem a largura e a altura do mesmo tamanho, 3.

Mas as matrizes podem ser muito mais simples, possuindo apenas uma linha ou uma coluna. Vamos ver então como podemos iniciar uma matriz:

```
var frutas = [ 'uva', 'maçã', 'tomate' ]; // sim tomate é uma fruta
```

E para acessarmos cada posição individualmente podemos ir diretamente pelo seu índice.

```
console.log(frutas[0]); // uva  
console.log(frutas[1]); // maçã  
console.log(frutas[2]); // tomate
```

Percebeu que sempre começamos em 0? Sim na maioria das linguagens a contagem sempre começa em 0, diferentemente do mundo *normal* onde começamos a contar em 1.

Para sabermos o tamanho do nosso *array* basta chamarmos a propriedade `length`.

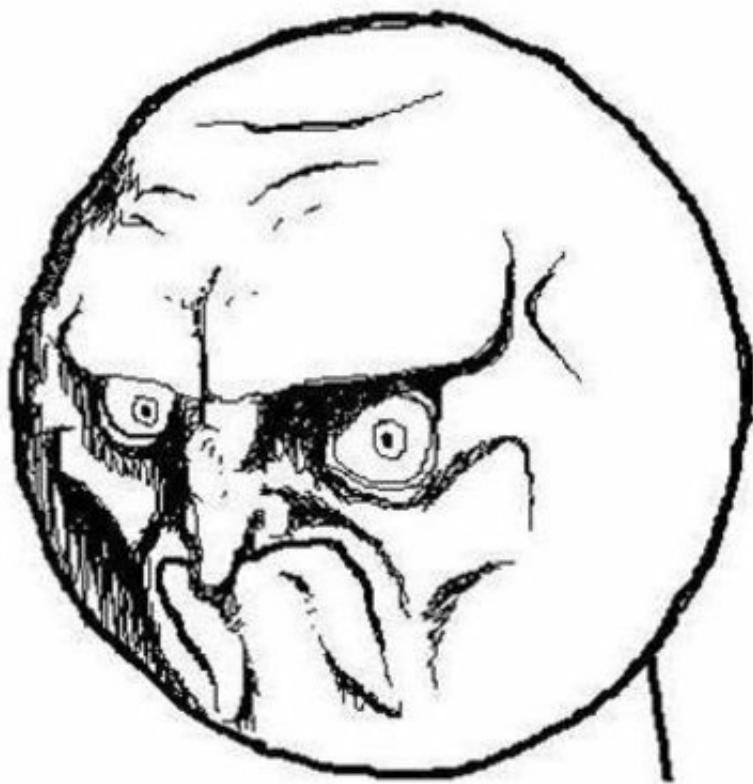
```
console.log(frutas.length); // 23
```

Iterando

Para iterar em um *array* temos algumas formas, veremos as mais utilizadas, como a função `forEach` que deve ser a mais utilizada para esse caso do *Array*. Outras formas de iteração veremos no módulo de lógica.

```
frutas.forEach( function(item) { console.log(item); } );
```

MUITO simples não?



NO.

Ok, então vamos analisar o que essa linha está fazendo, quando chamamos a função `forEach` ela irá executar uma função que passamos por parâmetro para **TODO** o item existente no `array`, ou seja para cada posição do `array` o `forEach` executa:

```
function(item) { console.log(item); }
```

E o que esse pedaço de código faz é receber em `item` o valor de cada posição do `array` e executar o código:

```
console.log(item);
```

Assim mostrando o valor de cada posição com `console.log`. Dessa vez ficou claro né?



[Exercício] Escreva um código onde você inicie um *array* com o nome de 5 dos seus amigos e depois faça ele mostrar a mensagem: "Eu gosto muito da(o) " + nome.

<div style="page-break-after: always;"></div>

Introdução à Lógica

O que é lógica de programação?

Lógica de Programação é a técnica de desenvolver sequências lógicas para atingir um determinado objetivo. Essas sequências lógicas são adaptadas para linguagem de computador pelo programador a fim de produzir um sistema. Essa sequência lógica é denominada algoritmo.

Sequência Lógica

```
Bata as claras em neve
Reserve
Bata bem as gemas com a margarina e o açúcar
Acrescente o leite e farinha aos poucos sem parar de bater
Por último agregue as claras em neve e o fermento
Coloque em forma grande de furo central untada e enfarinhada
Asse em forno médio, preaquecido, por aproximadamente 40 minutos
Quando espetar um palito e sair limpo estará assado
```



Sim a sequência lógica é como uma receita de bolo, você deve dizer passo-a-passo o que o computador deve fazer, também conhecido como **algoritmo** ele será seu guia para a solução de problemas.

Algoritmo

Um algoritmo é formalmente uma seqüência finita de passos que levam a execução de uma tarefa. Podemos pensar em algoritmo como uma receita, uma seqüência de instruções que dão cabo de uma meta específica. Estas tarefas não podem ser redundantes nem subjetivas na sua definição, devem ser claras e precisas. Como exemplos de algoritmos podemos citar os algoritmos das operações básicas (adição, multiplicação, divisão e subtração) de números reais decimais.

Até mesmo as coisas mais simples, podem ser descritas por seqüências lógicas. Por exemplo:

- Chupar uma bala.
- Pegar a bala
- Retirar o papel
- Chupar a bala
- Jogar o papel no lixo

[Exercício] Tendo esse conhecimento agora você deverá criar o seu algoritmo de tomar banho, no mínimo 4 e no máximo 10 passos.

É a partir do algoritmo que desenvolvemos o código em alguma linguagem de programação, já que para o algoritmo nós usamos uma linguagem mais natural, caso queiramos fazer um algoritmo mais acadêmico podemos utilizar o [Portugol](https://pt.wikipedia.org/wiki/Portugol) (<https://pt.wikipedia.org/wiki/Portugol>).

Vamos ver esse exemplo de conferir a maioridade:

```
inicio
    escreva("Qual sua idade?")
    leia(idade)
    se idade > 18 então
        escreva("Maior de idade")
    senão
        escreva("Menor de idade")
    fimse
fim
```

Com isso vemos que o Portugol possui uma sintaxe mais próxima à das linguagens de programação. Nesse módulo iremos conhecer exatamente quais são os comandos do JavaScript que vão nos auxiliar nessa tarefa.

Não vamos nos aprofundar nesse tema porém tenha em mente que você fará ele muitas vezes mentalmente, mas no início é melhor colocar o pensamento no papel antes de passar para linguagem de programação.

Instruções

Em informática uma instrução é a informação que indica a um computador uma ação elementar a executar, convém ressaltar que uma ordem/instrução isolada não permite realizar o processo completo, para isso é necessário um conjunto de instruções colocadas em ordem seqüencial lógica.

A instrução mais simples que temos no JavaScript é uma atribuição de valor. E isso significa o que?

Basicamente que quando criarmos uma variável e definirmos um valor para ela, estamos atribuindo a ela esse valor e para isso usamos o =:

```
// Exemplo  
var evento = "JS4Girls"
```

Nesse caso utilizamos a palavra var para criar uma variável nomeada de evento com o valor inicial igual a JS4Girls.

E por que nomeamos uma variável?



Porque sim não é resposta!

Mas então por que nomeamos uma variável?

É para que possamos utilizar o seu valor em outras partes do nosso programa. Sabendo disso vamos agora conhecer um tipo especial para essa variável, o tipo Boolean.

Boolean

Boolean é o nome desse tipo de variável, que é um tipo lógico, dado em homenagem da lógica booleana, George Boole. Esse tipo de lógica algébrica é simples de entender pois ela possuia apenas dois valores:

- verdadeiro / true / 1
- falso / false / 0

Você deve se perguntar, mas como assim funciona apenas com esses dois valores?

Você precisa pensar em abstrair o valor e transformar ele em verdadeiro ou falso.

Mas como assim?

Analisemos a seguinte afirmação:

Suissa tem 30 anos.

O que você pode abstrair de verdadeiro ou falso dessa afirmação?

Exatamente o que vimos no algoritmo de Portugol, que o Suissa é maior de idade. Basicamente é dessa forma que devemos pensar quando formos programar.

Um outro exemplo bem simples é:

Preciso ir para a Avenida Brasil e pergunto para o GPS onde estou e ele responde:

- Avenida Uruguai.

Nesse caso nossa abstração dá?

Falso!

Podemos analisar como um algoritmo:

```
local = leiaGPS()
se local <> "Avenida Brasil"
    retorno falso
senão
    retorno verdadeiro
```

Entendeu? No caso o símbolo `<>` é utilizado como **diferente**.

Vamos pegar uma situação real onde você pretende ir à praia. A decisão a ser tomada, então, é se a afirmação “vou à praia” será verdadeira ou falsa. Sabe-se que não é conveniente ir à praia com chuva. A condição, então, é estar ou não chovendo. Para tomar a decisão será preciso investigar a condição, verificando se a afirmação “Está chovendo” é verdadeira ou falsa. E como você somente irá à praia se NÃO estiver chovendo, a condição será tomada baseada na aplicação do operador NOT (que inverte o valor verdadeiro ou falso da resposta).

(vou à praia) = [NOT (está chovendo)]

Suponha que não esteja chovendo. Então:

(está chovendo) = (FALSO)

Portanto:

[NOT (está chovendo)] = [NOT (FALSO)] = VERDADEIRO

O que significa que o segundo termo da equação é VERDADEIRO. Substituindo este valor na equação e levando em conta a igualdade, teremos:

(vou à praia) = VERDADEIRO

Agora que entendemos o conceito de booleano vamos aprender as operações básicas que podemos fazer com esse valor.

AND / E

O **E** é um operador lógico que irá testar 2 premissas onde o retorno dela só será verdadeira **se todas as proposições forem verdadeiras**.

Exemplo:

Suissa é professor E Suissa é homem.

Se tivermos apenas uma proposição **falsa** toda a operação retornará o valor falso.

Então no E lógico **TODAS AS PROPOSIÇÕES PRECISAM SER VERDADEIRAS** para que ele seja verdadeiro.

Vejamos um exemplo real onde queremos ir ao cinema, porém para que isso aconteça precisamos ter dinheiro e estar de folga, então vamos montar a expressão:

[(estou de folga) AND (tenho dinheiro)]

Como primeiro termo é sempre a decisão a ser tomada. Logo, a equação lógica cuja solução decidirá se você irá ou não ao cinema será:

(vou ao cinema) = [(estou de folga) AND (tenho dinheiro)]

Para obter os resultados possíveis da operação AND vamos aplicar a ela as leis da lógica digital usando todas as combinações possíveis dos valores VERDADEIRO e FALSO (não vai dar muito trabalho, são apenas quatro). Aqui estão:

[FALSO] AND [FALSO] = FALSO

[FALSO] AND [VERDADEIRO] = FALSO

[VERDADEIRO] AND [FALSO] = FALSO

[VERDADEIRO] AND [VERDADEIRO] = [VERDADEIRO]

[Exercício] Monte uma operação lógica E com 3 premissas onde uma delas é falsa e as outras verdadeiras. E aplique os valores lógicos em cada proposição para mostrar seu resultado.

OR / OU

O OU é um operador lógico que irá testar 2 premissas onde o retorno dela só será verdadeira **se pelo menos uma proposição for verdadeira**.

Exemplo:

Suissa é contador OU Suissa é homem.

Se tivermos apenas uma proposição **verdadeira** toda a operação retornará o valor verdadeiro.

Então no OU lógico **PELO MENOS UMA PROPOSIÇÃO PRECISA SER VERDADEIRAS** para que ele seja verdadeiro.

(saio de casa) = (balada) OR (barzinho)

Nesse caso se eu receber um convite tanto para a balada como para o barzinho eu irei sair de casa, melhor se receber os 2 :p

[Exercício] Monte uma operação lógica OU com 3 premissas onde uma delas é falsa e as outras verdadeiras. E aplique os valores lógicos em cada proposição para mostrar seu resultado.

NOT / NEGAÇÃO

A operação de negação é exatamente o que você acha que ela faz, ela **NEGA** aquele valor e fazendo isso então ela o inverte.

No caso se você negar o falso ele se tornará verdadeiro e vice-versa.

~ Suissa é contador

Utilizando o ~ como negação temos na afirmação acima a seguinte resposta:

Suissa não é contador

(tomar banho) = [NOT (frio) AND (ter água)]

Com o operador NOT estou dizendo que só tomarei banho se não estiver frio. Mas nada que um aquecedor e duas toalhas não resolvam né? :p

[Exercício] Monte uma operação lógica OU com 3 premissas onde uma delas é falsa, uma é verdadeira e a outra pode ser qualquer valor porém utilizando o NOT. E aplique os valores lógicos em cada proposição para mostrar seu resultado.

Depois de conhecer essas operações básicas já podemos começar a testar nos proposições.

if / se

O if é a operação que irá testar nossos valores, como vimos anteriormente na parte do algoritmo. Agora vamos aprender a utilizá-la em conjunto com nossos valores booleanos.

A sintaxe para utilização do if é:

```
if( proposições ) {  
    // meu código  
}
```

Perceba então que o seu código dentro do if só será executado se o resultado das proposições for verdadeiro. É aí que aplicaremos nosso conhecimento anterior. Vamos analisar esse exemplo:

```
var idadeSuissa = 30;  
  
if(idadeSuissa > 18) {  
    console.log('MAIOR DE IDADE');  
}
```

Está lembrando desse código? Pois nós já o fizemos anteriormente:

```
inicio  
    escreva("Qual sua idade?")  
    leia(idade)  
    se idade > 18 então  
        escreva("Maior de idade")  
    senão  
        escreva("Menor de idade")  
    fimse  
fim
```

Claro que ainda falta a parte do senão, então vamos ver ela agora.

else / senão

O else é a negação do if, ou seja, quando o resultado das proposições for falso e o programa não entrar no bloco do if, ele entrará no bloco do else assim executando o código que ali estiver. Exemplo:

```
var idadeSuissa = 30;  
  
if(idadeSuissa >= 18) {  
    console.log('MAIOR DE IDADE');  
}  
else {  
    console.log('MENOR DE IDADE');  
}
```

[Exercício] Escreva um código que irá receber o ano que você nasceu em uma variável chama idade e irá testar se é MENOR que 1996, caso sim exiba a mensagem: "OK vc é de maior". Caso não, exiba: "Proibida entrada!"

else if

O else if nada mais é que mais um teste de if porém só é executado no else, ou seja, apenas se o primeiro if for falso.

```
var tempo = prompt("Que horas são?")
if (tempo < 13) {
    saudacao = "Bom dia";
} else if (tempo < 19) {
    saudacao = "Boa tarde";
} else {
    saudacao = "Boa noite";
}
```

Podemos notar então que se forem menos de 13 horas será Bom dia, se for mais, porém for menos de 19 horas será Boa tarde, se não será Boa noite. Simples não?

[Exercício] Escreva um código que irá receber o sexo de um cliente de uma boate e irá testar o sexo para definir o valor do ingresso. Se for mulher retorne: 15. Se não for mulher e for homem, retorne 25. Se não mostre a mensagem: "Sexo indefinido".

switch

Você deve ter percebido que agora nós começamos a ter mais testes lógicos e com o acréscimo deles nosso if começa a ficar sobrecarregado e quando isso acontece o switch vem para nos salvar.

O switch serve para testarmos várias condições e executar o código necessário, para entender melhor vamos analisar o código abaixo:

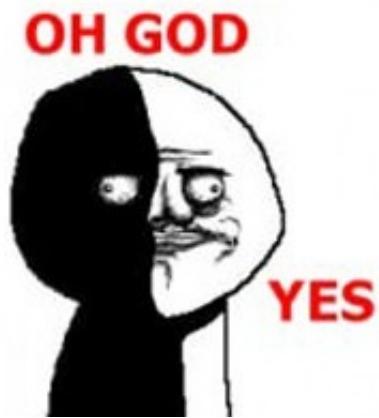
```
var estadoCivil = prompt("Qual seu estado civil?");

switch(estadoCivil) {
    case 'solteira':
        console.log("Bora pra festa!");
        break;
    case 'casada':
        console.log("Parabéns pelo casamento!");
        break;
    case 'divorciada':
        console.log("Deve ser um alívio!");
        break;
    case 'viúva':
        console.log("Meus pesames!");
        break;
    default: console.log("Complicado");
}
```

Nesse código testamos o valor que recebemos em estadoCivil com diferentes valores, como:

- solteira
- casada
- divorciada
- viúva

Utilizando a instrução default para executar um código quando nenhum dos outros cases foi verdadeiro. Dessa forma deixando nosso código mais simples e claro do que se colocássemos um monte de else if, não é?



E a partir do código logo notamos que a sintaxe para utilização do mesmo é:

```
switch( proposição ) {  
    case 'estado1':  
        // faz algo  
        break;  
    case 'estado2':  
        // faz algo  
        break;  
    case 'estado3':  
        // faz algo  
        break;  
    default: // faz outra coisa  
}
```

while

Já aprendemos que um algoritmo é uma sequencia de passos com objetivo de se chegar a um resultado. Em muitos casos precisamos repetir um certo "passo", para facilitar nossa vida temos os laços de repetição. Podemos repetir um bloco de comandos quantas vezes necessário.

A sintaxe para utilização do while é:

```
while( proposição ) {  
    //seu código  
}
```

Isso significa que o código ali dentro rodará toda vez que a proposição for verdadeira, vamos ver esse exemplo a seguir:

```
var numero = 1;  
  
while(numero <= 10) {  
    console.log(numero);  
    numero++;  
}
```

Nesse código estamos apenas mostrando os números de 1 até 10, onde a proposição `numero <= 10` é verdadeira apenas até a variável `numero` tiver o valor de 10, quando ela receber o valor 11 e chegar naquele teste novamente o programa irá sair desse loop(iteração).

[Exercício] Escreva um código onde inicie um número com o valor 0 e vá até 20, mostrando apenas os valores pares.

do while

Muito parecido com o `while`, porém com a diferença de que ele **sempre** irá executar o primeiro passo. Utilizando o mesmo código anterior, agora ficará:

```
var numero = 1;

do {
    console.log(numero);
    numero++;
} while(numero <= 10);
```

O do while é muito utilizado quando precisamos fazer certas checagens obrigatórias antes de começar alguma execução.

[Exercício]

for

O for é o loop mais comumente utilizado pois ele te "facilita" deixando mais claro o passo-a-passo da iteração. Em português nós dizemos:

para ... faça

Que significa:

```
for(inicialização; condição; expressão final) {
    // seu código
}
```

Vamos traduzir isso para o JavaScript:

```
for(var numero = 1; numero <= 10; numero++) {
    console.log(numero);
}
```

E pronto está dando o mesmo resultado que os exemplos dos outros loops, então basicamente é apenas uma forma diferente de escrever a mesma coisa.

Agora vamos analisar com mais calma o que está acontecendo na linha do for. Na primeira parte var numero = 1 estamos iniciando uma variável que será usada na nossa condição, logo após temos nossa condição numero <= 10 que será o teste que o for fará toda iteração para saber se ele entra para executar o código ou sai fora do loop. E finalmente a expressão numero++ que está incrementando, adicionando, 1 cada vez que o for é executado, fazendo assim o número subir toda iteração.

Mas também temos formas diferentes de escrever o mesmo for, confira logo abaixo:

```
var numero = 1;
for(; numero <= 10; numero++) {
    console.log(numero);
}
```

Nesse caso acima como já inicializamos a variável numero antes do for, não precisamos fazer isso novamente nele, por isso a primeira parte pode ficar vazia, agora veja o código abaixo:

```
for(var numero = 1; ; numero++) {
    if(numero > 10) break;
    console.log(numero);
}
```

No código acima estamos omitindo a nossa condição nesse caso o for ficará rodando para sempre, se não fosse o nosso teste interno if(numero > 10) break; que o faz sair(break) quando o numero for maior que 10. Percebeu que agora usamos a lógica inversa, pois antes testávamos se o numero era menor ou igual a 10 para que o for continuasse iterando.

Mas também podemos omitir as 3 partes:

```
var numero = 1
for(; ; ) {
    if(numero > 10) break;
```

```
console.log(numero);
numero++;
}
```

Dessa forma você precisa garantir que inicializou a variável fora do `for`, que vai testar a variável e dar o `break` no `for` quando necessário e não esquecer de incrementar a variável `numero`.

[Exercicio] Escreva um código que multiplique um número sempre pelo seu próximo, com valor máximo de 10, escreva utilizando as 4 formas apresentadas acima.

```
<div style="page-break-after: always;"></div>
```

Funções

Uma função é um conjunto de instruções utilizadas para executar uma determinada tarefa. Seu principal objetivo é evitar que um trecho de código seja repetido sempre que for preciso efetuar uma operação.

Imagine comigo que todo dia quando você acorda e vai escovar os dentes, precisa:

1. pegar escova de dente
2. pegar pasta de dente
3. abrir pasta de dente
4. colocar pasta de dente na escova
5. escovar os dentes
6. enxaguar a boca
7. cuspir

Agora imagine se em vez de você fazer esse passo-a-passo todo o dia pudesse apenas rodar uma função onde você inicia com os dentes sujos e sai com eles limpinhos.



Para isso serve uma função, toda vez que você repetir algum código, muito provavelmente ele pode ser encapsulado em uma função.

Sintaxe

Para criar uma função você usará o seguinte código:

```
function nome ( parametro ) {  
    //código a ser executado  
}
```

Assim como é usado a palavra reservada `var` para criar uma variável, a função também necessita de uma palavra reservada, a `function`.

Na primeira linha é criada uma função através da palavra `function`, em seguida temos o nome da função - que será usado para invocá-la. O nome da função pode ser qualquer palavra, que não seja uma [palavra reservada do JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar#Keywords) (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar#Keywords), e pode conter letras, dígitos, underlines e círões.

Após o nome da função temos parâmetros entre parênteses. O uso de parênteses indica que uma função está sendo usada e entre eles existe a palavra `parametro`, que será explicada [aqui](#).

O código a ser executado pela nossa função é escrito entre chaves. No nosso código as chaves começam no final da primeira linha e terminam na última linha. Tudo que estiver dentro destas chaves fará parte do escopo da função.

Parâmetros e Argumentos

Quando uma função é criada, no JavaScript, ela pode ou não receber dados que serão processados. Esses dados, no momento de criação da função, são chamados de *parâmetros*. Os parâmetros são variáveis que vão transmitir algum valor(argumento) para o código executado dentro da função. Se for necessário o uso de mais de um parâmetro na criação da função, eles devem ser separados por vírgula(,).

Os argumentos são os valores passados no momento de invocação da função, esses valores podem fazer parte de qualquer [tipo de dado](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Values,_variables,_and_literals#Valores) (https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Values,_variables,_and_literals#Valores).

Criando uma função de boas vindas com um parâmetro e uma função de multiplicar com 2 parâmetros:

```
//Função de boas vindas
function boasVindas (nome) {
    return "Eu estou feliz por você estar aqui, " + nome;
}

// Função que multiplica dois valores
function multiplicar (x, y) {
    return x * y;
}
```

Chamando a função de boas vindas passando um nome como argumento e a função multiplicar passando 2 números argumentos:

```
//Invocando a função boasVindas(nome)
boasVindas("Joana"); //Eu estou feliz por você estar aqui, Joana;

//Invocando a função multiplicar(x, y)
multiplicar(5, 2); //10
```

Uma função também pode receber o valor de uma variável como argumento. A função de boas vindas seria invocada da seguinte maneira:

```
//Criando a variável 'meuNome'
var meuNome = "Joana";
//Invocando a função boasVindas(nome)
boasVindas(meuNome); //Eu estou feliz por você estar aqui, Joana
```

Nota: Para passar uma string como um argumento direto para uma função, como em `boasVindas("Joana")`, é necessário que este valor esteja entre aspas. Se essa função receber como argumento um valor de texto sem aspas, a função vai interpretar o argumento como uma variável, podendo gerar erros inesperados.

[Exercício] Crie uma função que calcule o quadrado de um número. Não precisa retornar um valor, apenas calcular.

Retorno

Uma função pode ser criada para efetuar cálculos ou executar rotinas. Quando o objetivo de uma função não é executar uma rotina, mas sim processar dados, é comum que ela faça um retorno. No JavaScript esse retorno é dado pela palavra

return seguida pelo dado que ela vai retornar.

No exemplo anterior o return foi usado para retornar o valor da multiplicação do parâmetro x pelo parâmetro y.

```
...
return x * y;
...
```

Invocando uma função

Uma vez que a função já foi criada, ela pode ser invocada da seguinte maneira:

```
nomeDaFuncao(parametro1, parametro2);
```

[Exercício] Crie uma função para calcular o IMC e invoque-a passando dois argumentos. *Cálculo do IMC: peso / (altura * altura)*

Uma função também pode ter seu valor de retorno associado à uma variável. Da seguinte forma:

```
var resultado = nomeDaFuncao(parametro1, parametro2);
```

Vamos re-utilizar um exemplo passado, de multiplicar, agora dessa forma:

```
var multiplicar = function (x, y) {
  return x * y;
}
```

Além disso nós também podemos passar uma função como argumento, assim como podemos também retornar uma função.

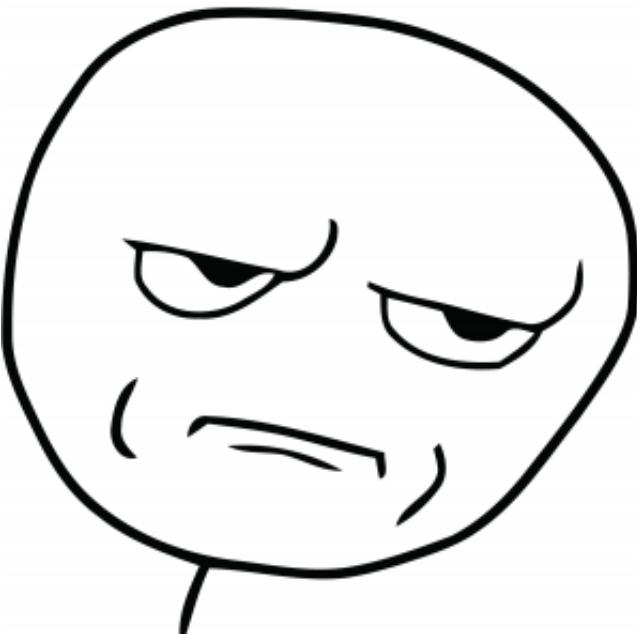
Porém como esse assunto já sai do escopo dessa primeira aula eu lhe aconselho a posteriormente fazer o [Curso de JavaScript Funcional GRATUITO](#) (<https://github.com/Webschool-io/workshop-js-funcional-free>) para maior entendimento sobre isso.

[Exercício] Crie duas variáveis, uma para armazenar o valor do peso e a outra para altura. Feito isso, crie uma função para calcular o IMC e invoque-a passando as duas variáveis criadas como argumentos. Associe o retorno dessa função à uma variável chamada resultadoIMC

```
<div style="page-break-after: always;"></div>
```

Objetos

O JavaScript é orientado a objetos.



- Mas o que é um objeto?

Um objeto é uma coleção de propriedades.

- E o que é uma propriedade?

Propriedade é uma associação entre um nome e um valor, isso mesmo, é simplesmente uma ligação. Um valor de propriedade também pode ser uma função, que é então considerada um método daquele objeto. Além dos objetos que são pré-definidos da linguagem, você pode definir seus próprios objetos de forma independente para resolver a necessidade de seu projeto.

Visão geral de objetos

Objetos podem ser comparados com objetos na vida real. Seu conceito pode ser entendido com objetos tangíveis da vida real.

Em JavaScript, um objeto é uma entidade independente, com **propriedades** e **tipo**. Compare-o com um carro, por exemplo. Um carro é um objeto, porém, o mesmo possui várias propriedades. Um carro tem uma cor, um modelo, peso, marca, tipo de combustível, etc. De mesma forma, objetos em JavaScript podem ter propriedades, que definem suas características.

![Exemplo](https://raw.githubusercontent.com/Webschool-io/js4girls/master/material-didatico/images/objects-img1.jpg)

Objetos e propriedades

Um objeto em JavaScript tem propriedades associadas a ele. Uma propriedade de um objeto pode ser explicada como uma variável que é ligada ao objeto. Propriedades de objetos são basicamente as mesmas que variáveis normais em JavaScript, exceto pelo fato de estarem ligadas a objetos. As propriedades de um objeto definem as características do objeto. Além disso cada propriedade de um objeto pode ter propriedades também. Por exemplo, no objeto carro podemos ter a propriedade motor. Mas a propriedade motor pode ter várias propriedades, como potencia, tipo combustível, etc. Mas não se preocupe, vamos ver mais sobre propriedades de propriedades mais abaixo.

Você acessa as propriedades de um objeto com uma simples notação de ponto:

```
nome_do_objeto.nome_da_propriedade
```

Não se esqueça que o JavaScript é **case sensitive**, tanto o nome do objeto quanto um nome de propriedade diferem em maiúsculas/minúsculas (por exemplo, cor e Cor são propriedades diferentes). Você pode definir uma propriedade atribuindo um valor a ela. Por exemplo, vamos criar um objeto chamado meuAviao e dar a ele propriedades chamadas modelo, fabricante e ano, conforme mostrado a seguir:

```
var meuAviao = new Object();
meuAviao.fabricante = "Airbus";
meuAviao.modelo = "A380";
```

```
meuAviao.ano = 2012;
```

Propriedades de objetos também podem ser acessadas ou alteradas usando-se notação de colchete. O código abaixo mostra como usar colchetes:

```
meuAviao[ "fabricante" ] = "Airbus";
meuAviao[ "modelo" ] = "A380";
meuAviao[ "ano" ] = 2012;
```

[Exercicio]

Crie um objeto e chame ele de "MeuVestido". Em seguida crie as seguintes propriedades: "Cor", "Tamanho", "Marca" e "Tipo" (Curto ou longo).

Criando novos objetos

Além de alguns objetos predefinidos (Date, Array, String) e objetos predefinidos do browser (Window, Document, Navigator) você pode criar seus próprios objetos. Não se preocupe com os objetos predefinidos que eles serão vistos mais adiante. Existem algumas formas de você criar um objeto, pode ser utilizando um inicializador de objeto, utilizando uma função construtora ou utilizando o método Object.create. Vamos ver os três logo abaixo:

Usando inicializadores de objeto

O uso de inicializadores de objeto é às vezes conhecido como criar objetos com notação literal. A sintaxe para um objeto usando-se um inicializador de objeto é:

```
var meuObjeto = {propriedade1: "valor_da_propriedade1", propriedade2: "valor_da_propriedade2",
propriedade3: "valor_da_propriedade3"};
```

Repare que utilizamos 3 propriedades, mas poderíamos ter utilizando N propriedades se fosse necessário. E tem mais um detalhe show de bola, uma propriedade pode ter suas próprias propriedades. Veja que o exemplo abaixo cria o objeto minhaHonda com três propriedades. Note que a propriedade motor é também um objeto com suas próprias propriedades.

```
var minhaHonda = {cor: "rosa", rodas: 2, motor: {cilindros: 3, potencia: 125, combustivel:
"Gasolina"}};
```

Usando uma função construtora

Alternativamente, você pode criar um objeto com estes dois passos:

- Defina o tipo de objeto escrevendo uma função construtora. Há uma forte convenção, e com boa razão, de se usar uma letra inicial maiúscula.
- Crie uma instância do objeto com *new*.

Para definir um tipo de objeto, crie uma função para o tipo de objeto que especifique seu nome, suas propriedades e seus métodos. Por exemplo, suponha que você criar um tipo objeto para carros. Você quer que esse tipo de objeto seja chamado carro, e você quer ele tenha propriedades de marca, modelo e ano. Para fazer isto, você escreveria a seguinte função:

```
function Carro(marca, modelo, ano) {
  this.marca = marca;
  this.modelo = modelo;
  this.ano = ano;
}
```

Note o uso de *this* para atribuir valores às propriedades do objeto com base nos valores passados para a função.

Agora você pode criar um objetos simplesmente chamando a função meucarro como se segue:

```
var meucarro = new Carro("Eagle", "Talon TSi", 1993);
var meucarro2 = new Carro("Corolla", "Xi", 2014);
```

Usando o método Object.create

Esse terceiro método não será apresentado pois não haverá tempo suficiente para a conclusão da apresentação. Mas nesse [Link \(https://msdn.microsoft.com/pt-br/library/ff925952\(v=vs.94\).aspx\)](https://msdn.microsoft.com/pt-br/library/ff925952(v=vs.94).aspx) é possível entender como utilizar a função

Object.create. Dever de casa hein, tem que fazer mesmo, pois além de aprender o método é possível ver como funcionam as documentações das linguagens e vocês já vão se familiarizando com o formato.

Typeof

É um método nativo do Javascript, se formos ler a documentação vamos encontrar isso: "O operador typeof retorna uma string indicando o tipo de um operando." Simplesmente vai dizer o que é aquilo, um número, um objeto, uma array... Parece simples né? Infelizmente não é tão simples assim, vamos lá!

O comportamento esperado com o Operador typeof seria o retorno de uma String indicando o tipo do Operando avaliado. Entretanto, o typeof tem um comportamento "inesperado" no JavaScript, sendo quase uma bruxaria! Uma mística em torno do seu poder que vamos decifrar ainda hoje:

```
// Crie essas variáveis ou Operandos
var teste1 = null;
var teste2 = [1, 2, 3, 4, 5, 6];
var teste3 = new Number(10);
var teste4 = 10;
var teste5 = new Array(1, 2, 3, 4, 5);
var teste6 = 'TutsMais';

// Vamos Efetuar os testes com o console.log()
console.log(typeof teste1);
console.log(typeof teste2);
console.log(typeof teste3);
console.log(typeof teste4);
console.log(typeof teste5);
console.log(typeof teste6);
```

Veja que a nossa saída será:

```
object // O null é um objeto?????? Oo
object // Eita caramba, não seria um Array?
object // Perfeito, 10 é um número
number // Perfeito também
object // Eita, não seria uma Array?
string // Perfeito
```

Calma que não é nenhuma magia negra não! Fiquem calmas! Realmente comportamento do typeof está correto, realmente ele deve retornar isso, como descrito no ECMAScript que prevê este resultado. Então tome MUITO CUIDADO ao usar o typeof em suas condições e estruturas da linguagem, o resultado realmente pode ser inverso a sua lógica e o seu código se perder na magia do JavaScript. A maioria dos Operadores no JavaScript porta-se de forma esperada, o problema do typeof é que o mesmo possui um comportamento "Exótico". O operador typeof espera um valor e devolve uma das strings: Number, String, Boolean, undefined, function ou Object. Por isso fica fácil se confundir se ele não for bem entendido. Mas então, o que usar? Vamos usar o instanceof!

Instanceof

No uso do instanceof nós vamos pedir se determinado objeto é um número, array, etc e a função simplesmente vai retornar sim(true) ou não(false)! A sintaxe da função é a seguinte:

```
objeto_a_ser_testado instanceof tipo-constructor
```

Vejamos um exemplo:

```
var teste3 = [2, 3, 5, 1, 2, 3];
console.log(teste3 instanceof Array);
```

Ou seja, no exemplo acima estamos pedindo se o objeto 'teste3' é uma Array. Nesse caso a função instanceof irá retornar true. Sim, teste3 é um Array. Esse é mais fácil né? Sem resultados inesperados!

[Exercício]

Sem executar o código, qual será a resposta do instanceof em cada caso abaixo?

```
var myDate = new Date();
myDate instanceof Date; // Vai retornar verdadeiro ou falso?
myDate instanceof Object; // Vai retornar verdadeiro ou falso?
myDate instanceof String; // Vai retornar verdadeiro ou falso?
myDate instanceof Number; // Vai retornar verdadeiro ou falso?
```