



JavaScript

Introdução ao DOM

Prof.: MSc. Arcanjo Miguel Mota Lopes

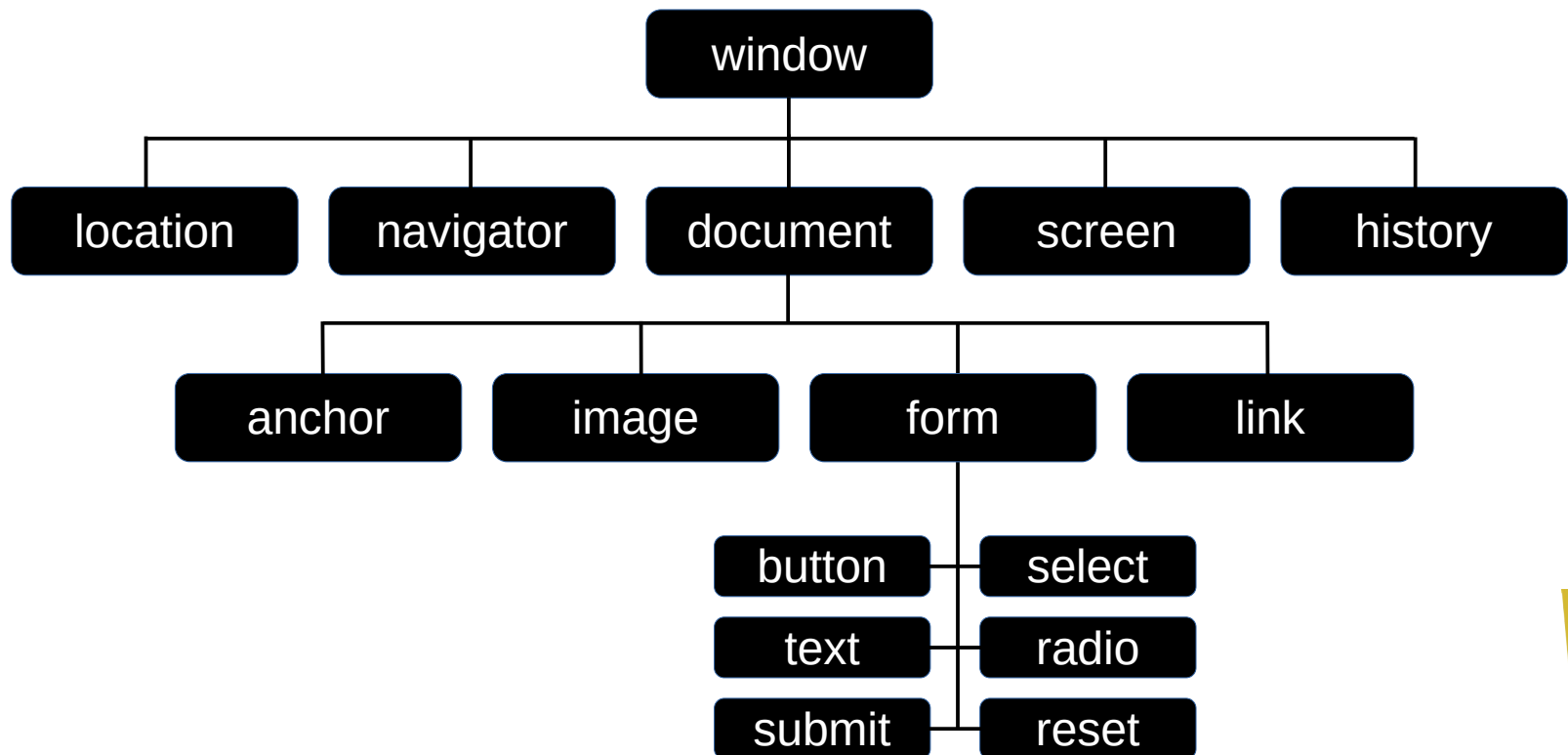
As 3 camadas

- Conforme já visto, o desenvolvimento client-side é baseado em 3 camadas principais:
 - **Conteúdo**, viabilizado pelo **HTML**
 - **Estilo**, viabilizado pelo **CSS**
 - **Comportamento**, viabilizado pelo **JavaScript**
- Nesta primeira parte do curso, iremos abordar a terceira camada: **o Comportamento**



Estrutura DOM

- Quando um browser carrega uma página, ele cria uma hierarquia de objetos para representar essa página

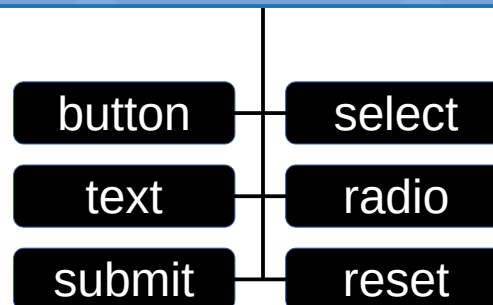


Estrutura DOM

- Quando um browser carrega uma página, ele cria uma hierarquia de objetos para representar essa página

Essa hierarquia de objetos é chamada **DOM** (Document Object Model) e ela fica armazenada em memória

A DOM possui uma ligação viva com o conteúdo da página, de forma que mudanças neste modelo são imediatamente refletidas na página Web



Propriedades do DOM

- **Window:** é o objeto do navegador que está sempre no topo da hierarquia. Ele é criado automaticamente pelo navegador.
- **Document:** possui várias propriedades que se referem a outros objetos que permitem o acesso e modificação do conteúdo da página web.
- **Form:** É representado por tags de *form*.



Propriedades do DOM

- **Link:** É representado por tags de *link*.
- **Anchor:** É representado por tags de *href*.
- **Form Control Elements:** pode ter muitos elementos de controle, como campos de texto, botões, botões de rádio, caixas de seleção, etc.



Objeto **window**

- O objeto **window** representa uma aba do browser, e é o nó raiz do **Document Object Model** (DOM)
- Tecnicamente, toda variável ou função de escopo global pertence ao objeto **window**
 - Desta forma, as duas sequências de códigos abaixo são similares:

```
document.write("UFAM");  
alert("Hello ");  
unidade = "IComp";
```

```
window.document.write("UFAM");  
window.alert("Hello");  
window.unidade = "IComp";
```



Objeto **window**

- Algumas propriedades e métodos do objeto **window**:
 - `alert()`, `prompt()`, `confirm()`
 - `open()` e `close()`
 - `setTimeout()` e `setInterval()`
 - `navigator`, `document` e `screen`
 - `location` e `history`



Objeto **window**

- **Navigator:** contém informações sobre o browser.

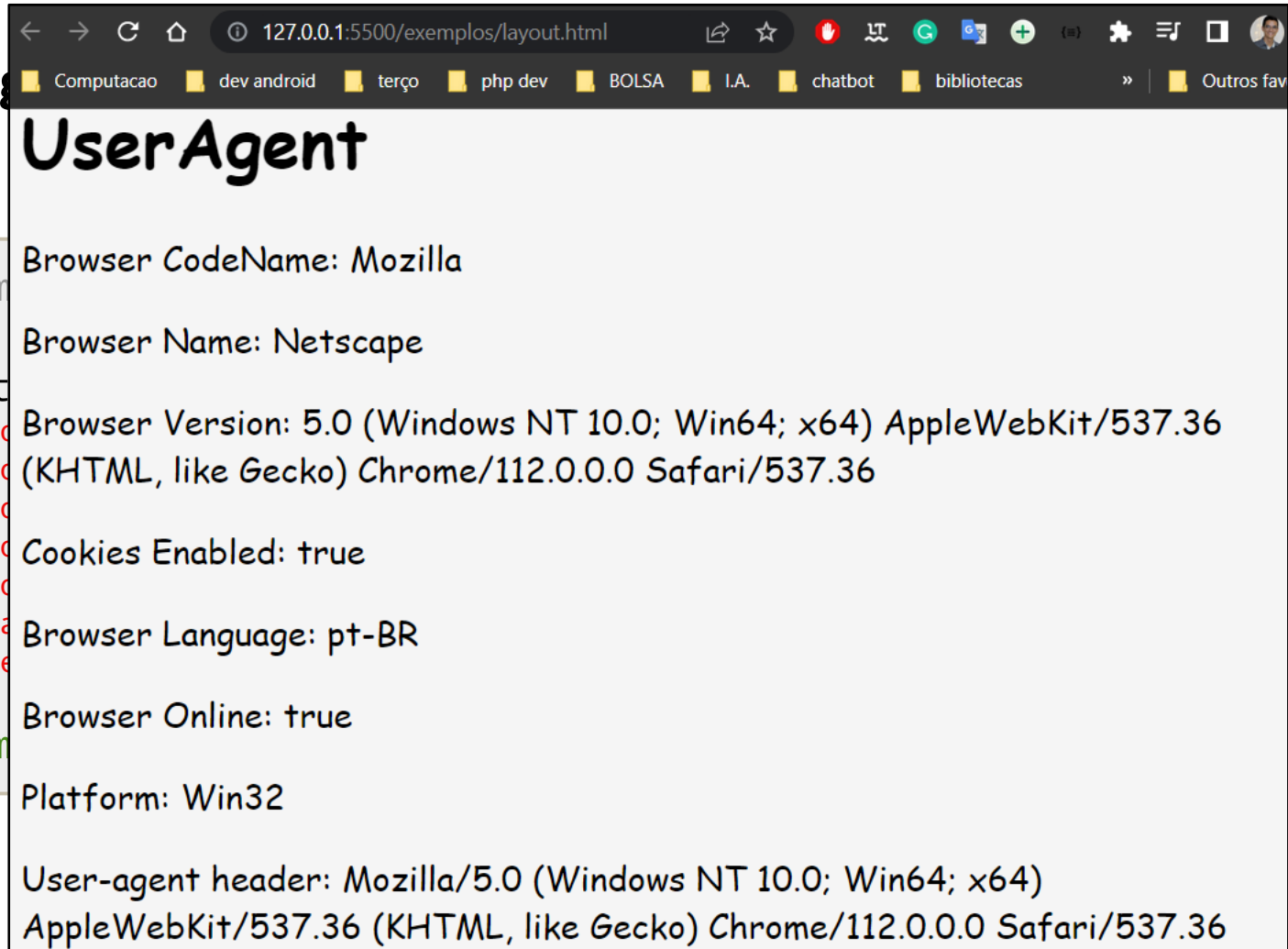
```
#Example JS window.navigator.userAgent
```

```
let text = "<p>Browser CodeName: " + window.navigator.appCodeName + "</p>" +  
"<p>Browser Name: " + window.navigator.appName + "</p>" +  
"<p>Browser Version: " + window.navigator.appVersion + "</p>" +  
"<p>Cookies Enabled: " + window.navigator.cookieEnabled + "</p>" +  
"<p>Browser Language: " + window.navigator.language + "</p>" +  
"<p>Browser Online: " + navigator.onLine + "</p>" +  
"<p>Platform: " + window.navigator.platform + "</p>" +  
"<p>User-agent header: " + window.navigator.userAgent + "</p>";  
  
document.writeln(text);
```



Objeto **window**

- Navig



The screenshot shows a web browser window with the address bar displaying "127.0.0.1:5500/exemplos/layout.html". The browser's toolbar includes various icons for navigation and development. Below the toolbar, a list of bookmarks is visible, including "Computacao", "dev android", "terço", "php dev", "BOLSA", "I.A.", "chatbot", "bibliotecas", and "Outros fav". The main content area of the browser displays the title "UserAgent" in a large, bold font. Below the title, a list of browser properties is shown, including "Browser CodeName: Mozilla", "Browser Name: Netscape", "Browser Version: 5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36", "Cookies Enabled: true", "Browser Language: pt-BR", "Browser Online: true", "Platform: Win32", and "User-agent header: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36".

Browser CodeName: Mozilla

Browser Name: Netscape

Browser Version: 5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36

Cookies Enabled: true

Browser Language: pt-BR

Browser Online: true

Platform: Win32

User-agent header: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36



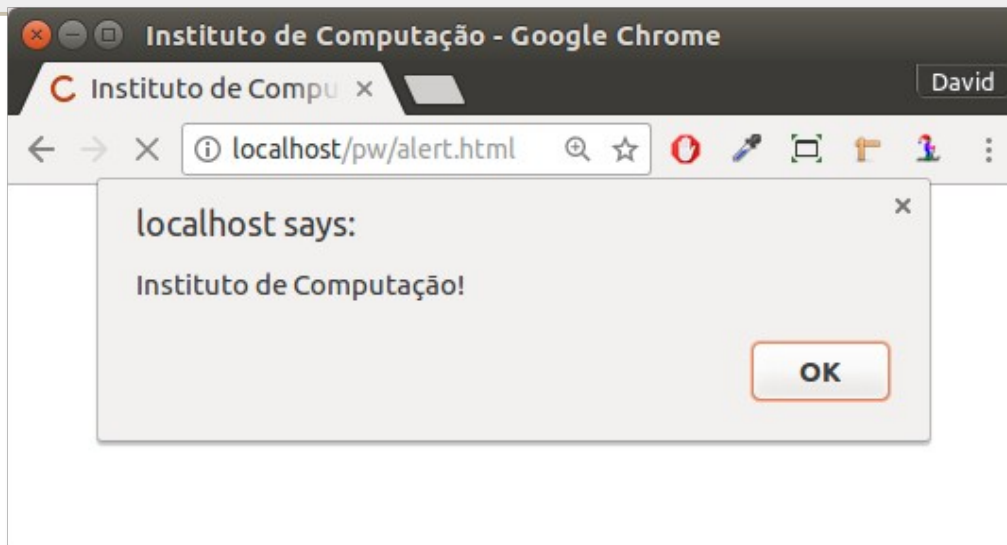
Alert, Confirm e Prompt

- Interação com usuários usando caixas de diálogo

`window.alert(msg) ;`

- Mostra uma mensagem em uma caixa que é fechada quando o usuário clica no botão OK

```
window.alert("Instituto de Computação!")
```

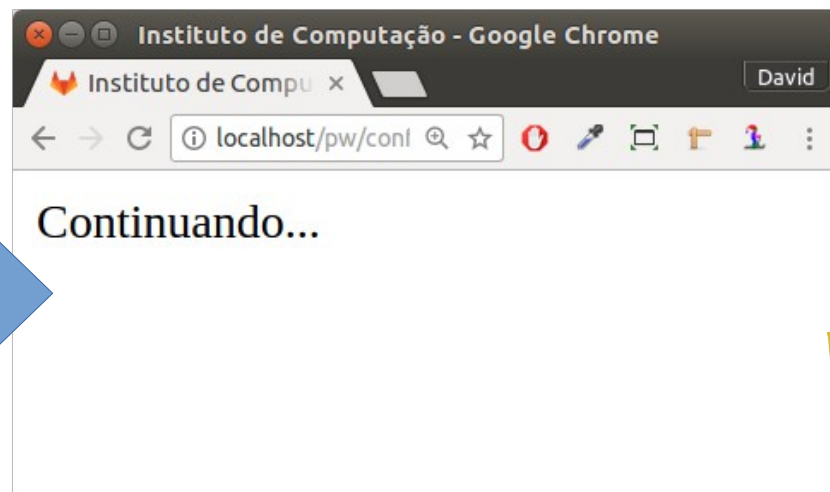
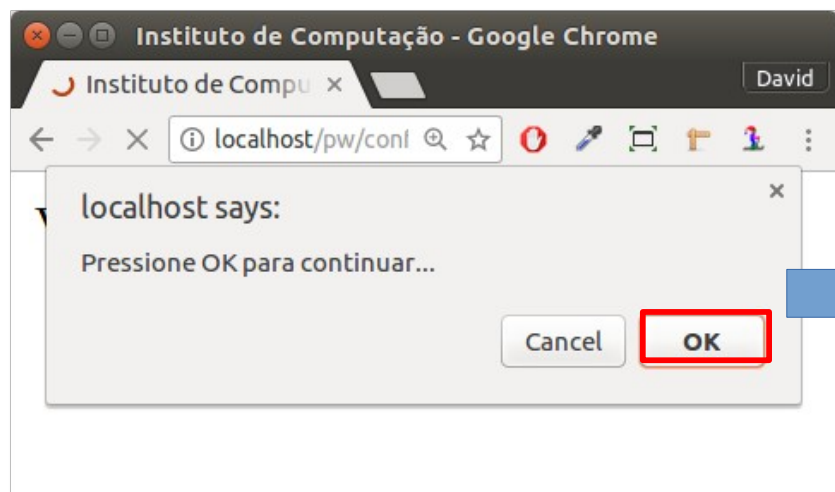


Alert, Confirm e Prompt

```
let r = window.confirm(msg);
```

- Mostra uma mensagem junto com os botões OK e Cancel.
- Retorna **true** se o usuário clica em OK, e **false** caso contrário

```
let r = window.confirm("Pressione OK para continuar...");  
if (r) document.writeln("Continuando...");
```

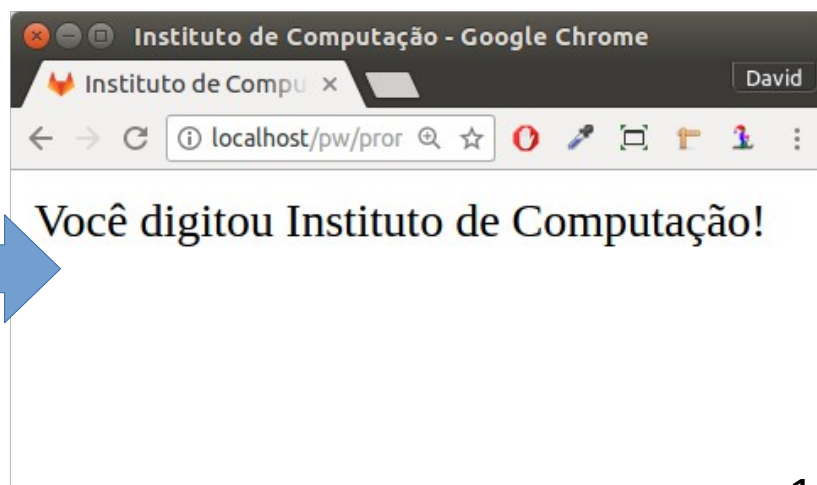
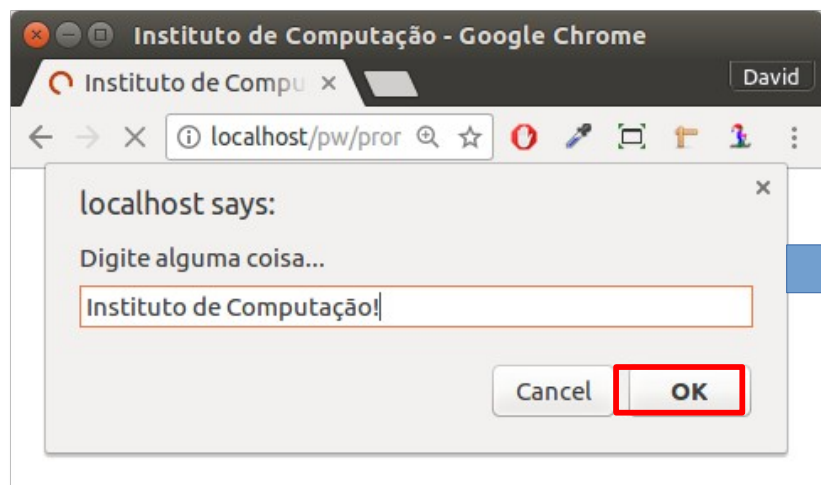


Alert, Confirm e Prompt

```
let r = window.prompt(msg, default) ;
```

- Mostra uma mensagem e um input de dados
- O método retorna o dado informado pelo usuário, ou **null** caso o botão **Cancel** seja clicado

```
let dado = window.prompt("Digite alguma coisa...");  
if (dado) document.writeln("Você digitou " + dado);
```



DOM (Document Object Model)

- É uma interface (API) independente de plataforma e linguagem que permite aos programas e scripts atualizarem dinamicamente a estrutura, conteúdo e estilo de documentos XML e HTML.
- Não é uma linguagem de programação, mas sem ele as linguagens não teria nenhum modelo ou noção de páginas web, documentos HTML, documentos SVG e suas partes componentes.



DOM (Document Object Model)

- É uma interface (API) independente de plataforma e linguagem que permite aos programas e scripts atualizarem dinamicamente a estrutura, conteúdo e estilo de documentos XML e HTML.

```
#Python DOM exemplo
import xml.dom.minidom as m
doc = m.parse(r"C:\Projects\Py\ex1.xml")
doc.nodeName #Propriedade DOM para Document Object
p_list = doc.getElementsByTagName("intro")
```

```
#JavaScript DOM exemplo
<script>
const element = document.getElementById("intro").innerHTML;
    alert("Primeiro texto é " + element)
</script>
```



Estrutura DOM

- O termo 'Modelo Estruturado' às vezes é usado para descrever a representação em árvore de um documento.
- Cada ramificação da árvore termina em um nó, e cada nó contém objetos 'escultado' por eventos que podem ser adicionados aos nós e acionados na ocorrência de um determinado evento.



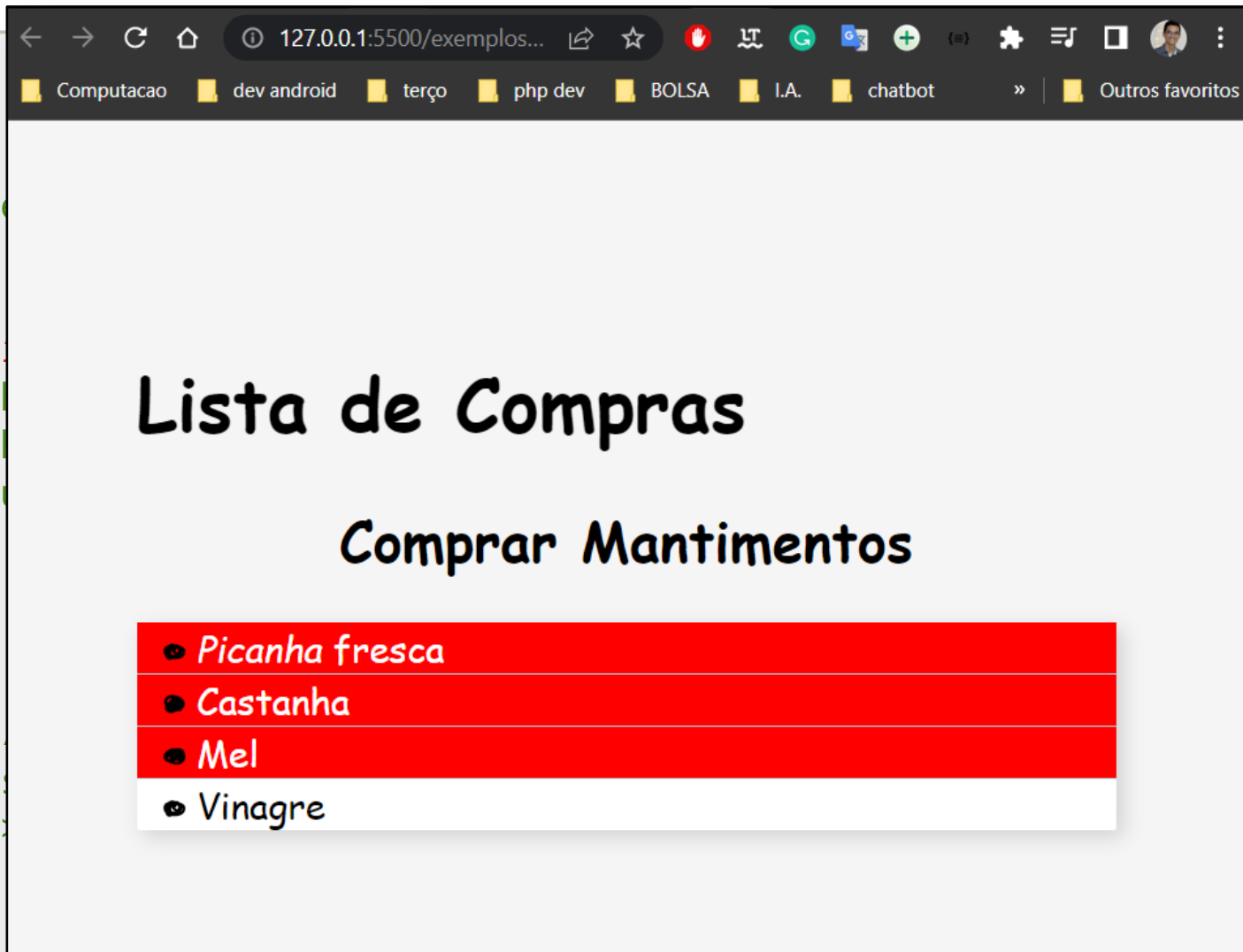
Estrutura DOM

```
<!DOCTYPE html>
<head>
  <title>Lista de Compras</title>
</head>
<body>
  <div id="page">
    <h1 id="header">Lista de Compras</h1>
    <h2>Comprar Mantimentos</h2>
    <ul>
      <li id="one" class="hot"><em>Picanha</em> fresca</li>
      <li id="two" class="hot">Castanha</li>
      <li id="three" class="hot">Mel</li>
      <li id="four">Vinagre</li>
    </ul>
    <script src="js/list.js"></script>
  </div>
</body>
</html>
```



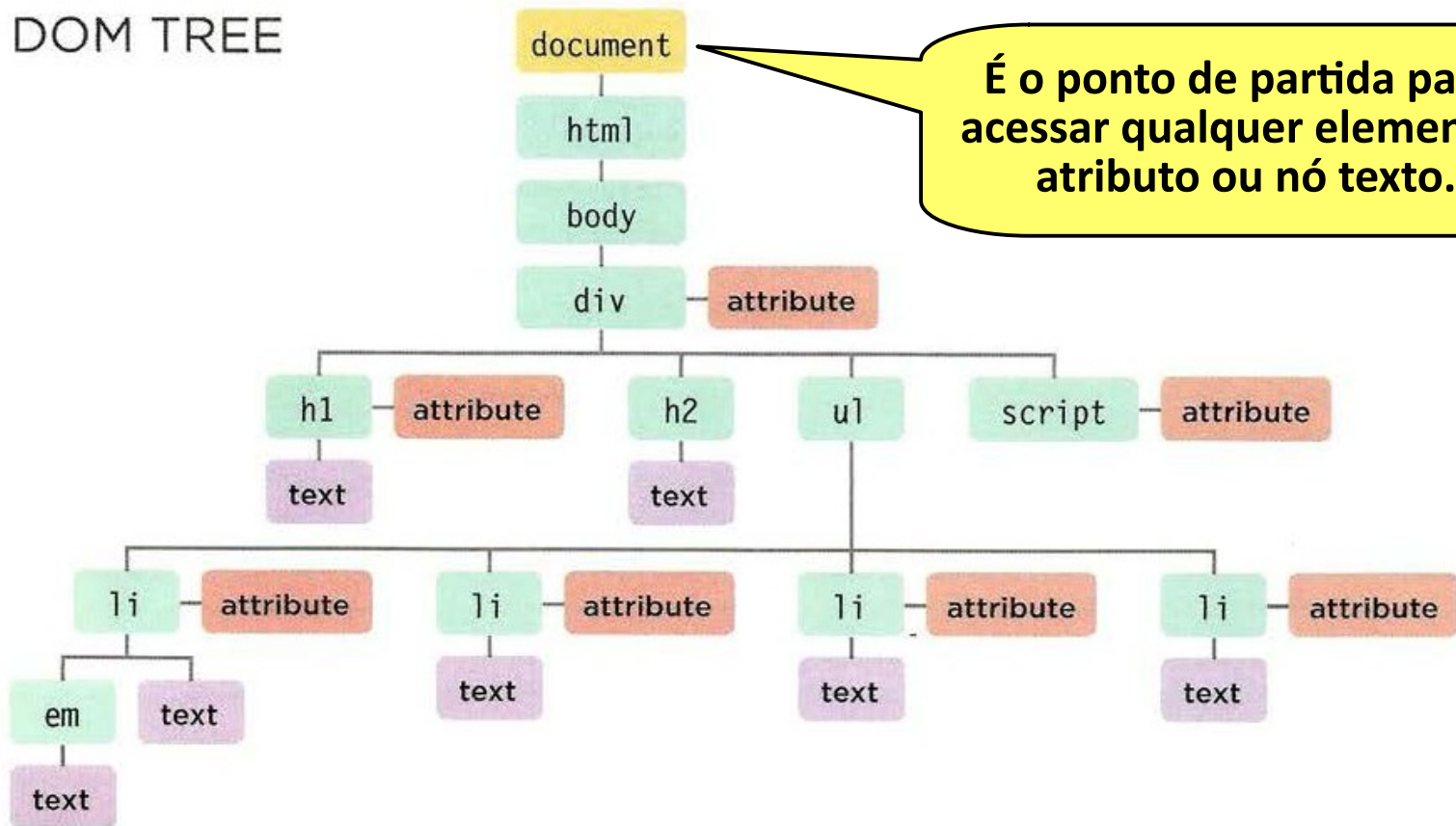
Estrutura DOM

```
<!DOCTYPE  
<head>  
  <titl  
</head>  
<body>  
  <div  
    <  
    <  
    <  
    <  
  </div>  
</body>  
</html>
```



Estrutura DOM

DOM TREE

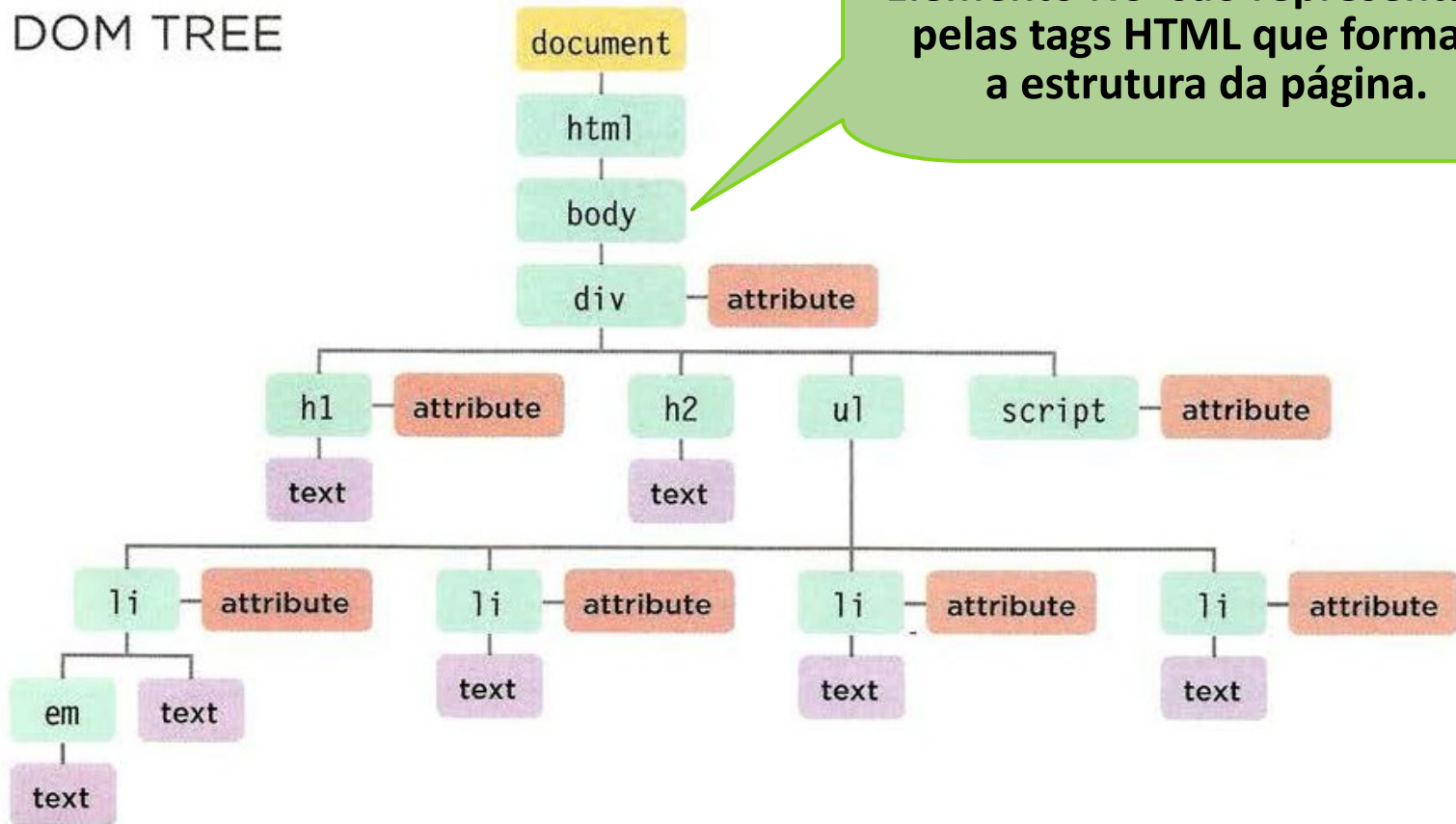


É o ponto de partida para acessar qualquer elemento, atributo ou nó texto.



Estrutura DOM

DOM TREE

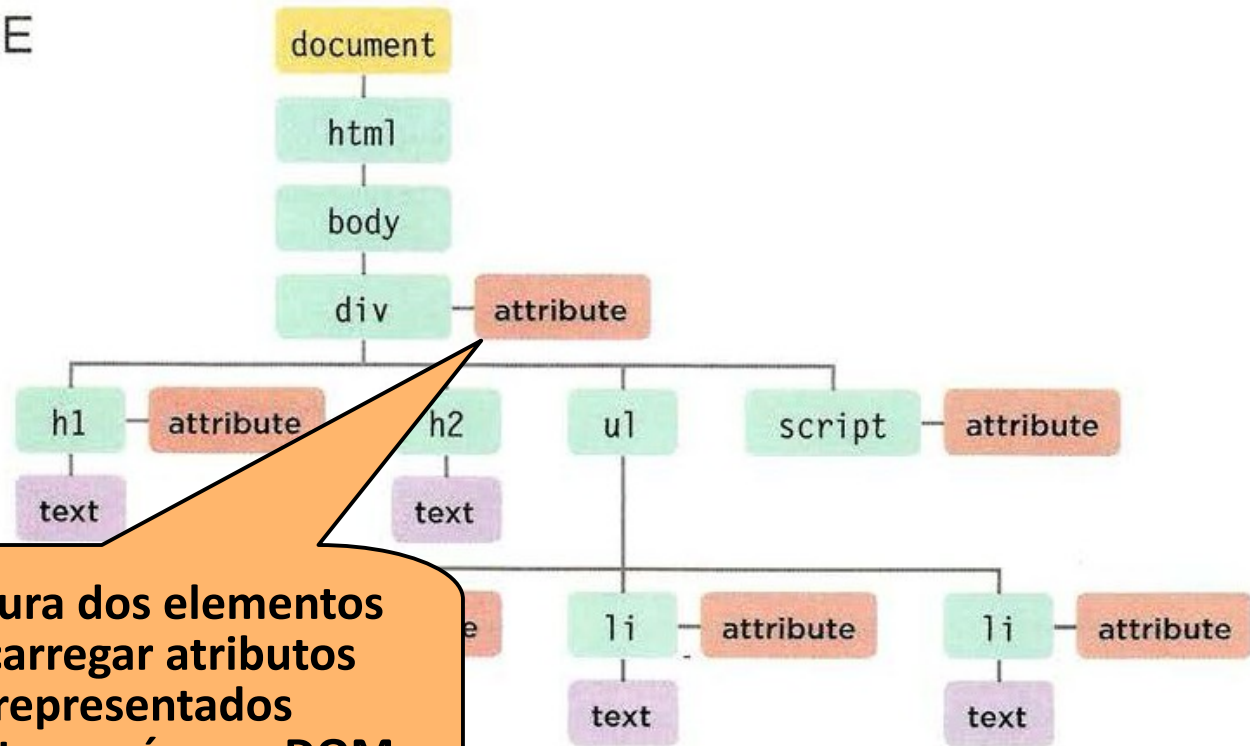


'Elemento Nó' são representados pelas tags HTML que formam a estrutura da página.



Estrutura DOM

DOM TREE

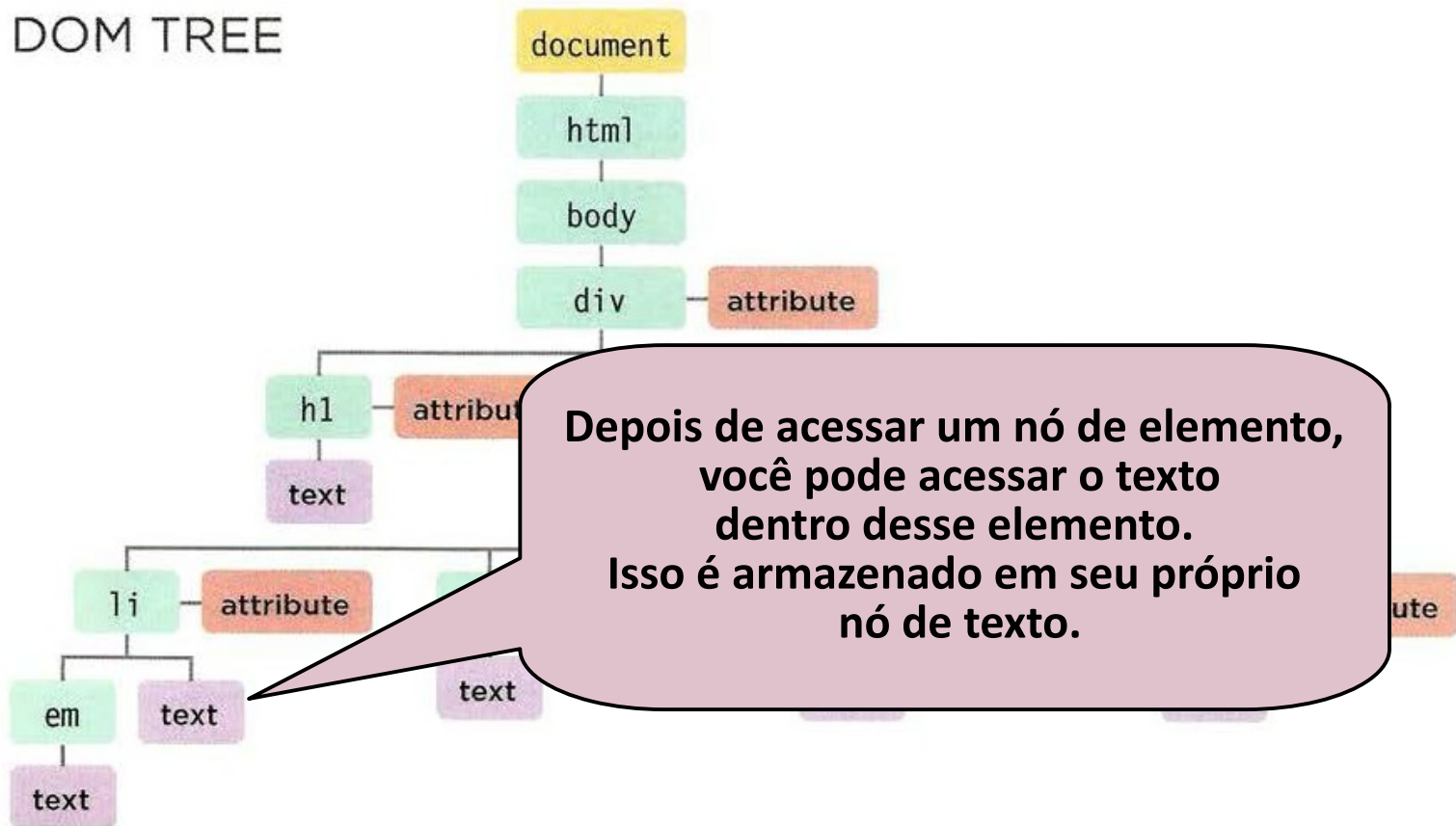


As tags de abertura dos elementos HTML podem carregar atributos e estes são representados por nós de atributos na árvore DOM. Exemplo: id, class e etc.



Estrutura DOM

DOM TREE



Objeto **document**

- O objeto **document** representa a página carregada em uma aba, e é o ponto de acesso aos elementos da página

```
<html>
  <head>
    <title>Instituto de Computação</title>
  </head>
  <body>
    <h1>Instituto de Computação</h1>

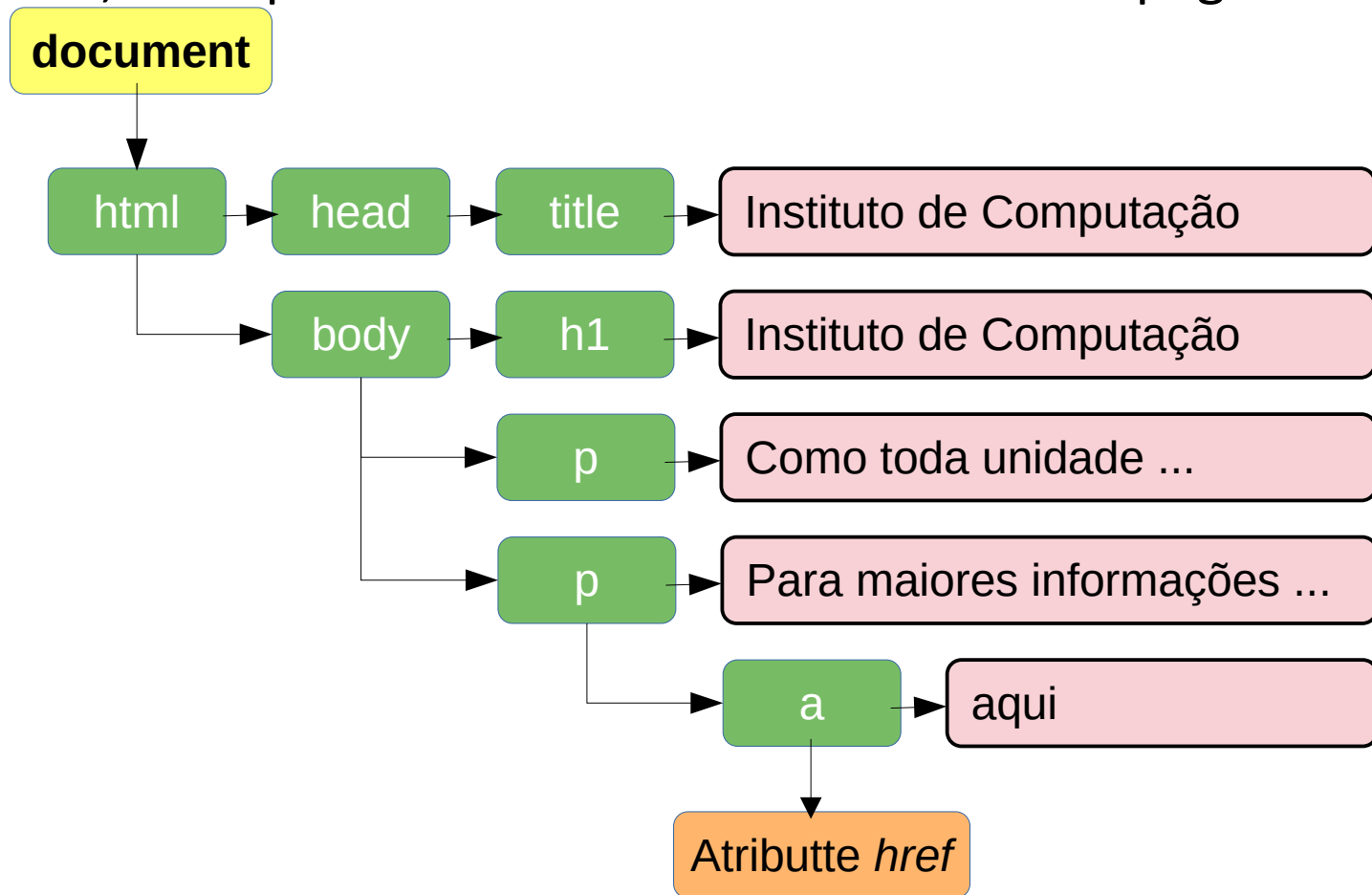
    <p>Como toda unidade acadêmica, o Instituto de
      Computação atua no ensino, pesquisa e extensão.
    </p>

    <p>Para mais informações sobre o Icomp, clique
      <a href="http://icomp.ufam.edu.br">Aqui</a>
    </p>
  </body>
</html>
```



Objeto **document**

- O objeto **document** representa a página carregada em uma aba, e é o ponto de acesso aos elementos da página



Estrutura DOM

- **Atividade:** Com base no Slide anterior, construa árvore DOM, considerando apenas os elementos, sem os atributos

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset=UTF-8">
    <title>Demo</title>
    <script src="meu_arquivo_javascript.js"></script>
  </head>
  <body>
    <h1 id="id_h1" class="classe_h1">Sou um cabeçalho!</h1>
    <p id="id_p" class="classe_p">
      Um texto qualquer dentro de uma tag de parágrafo. Aqui também
      temos outras tags, como <a href="#">um link</a>, ou um texto
      <b>em negrito</b>.
    </p>
    <p id="id_p" class="classe_p">
      Este é outro parágrafo.
    </p>
  </body>
</html>
```



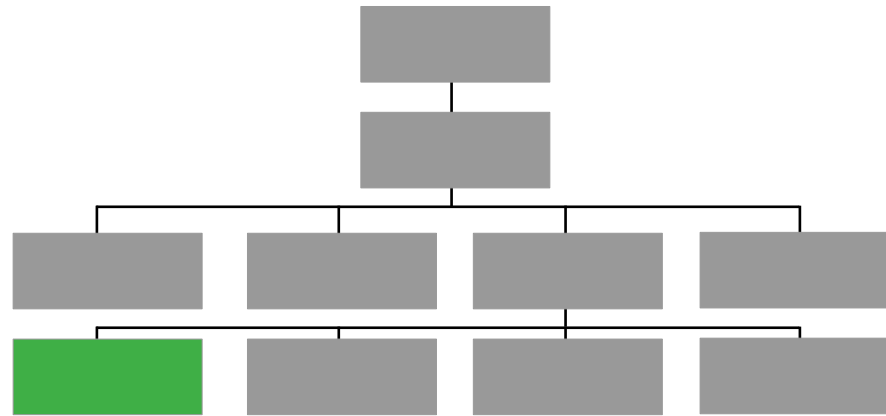
Acessando os Elementos da Árvore DOM

- Para acessar os elementos da árvore DOM, envolve dois passos:
 1. Localizar o nó que representa o elemento do qual deseja trabalhar.
 2. Utilizar o conteúdo Texto, Elemento Filho, e/ou Atributos.



Acessando os Elementos

- Acessando um elemento individual.



- **`getElementById()`** :

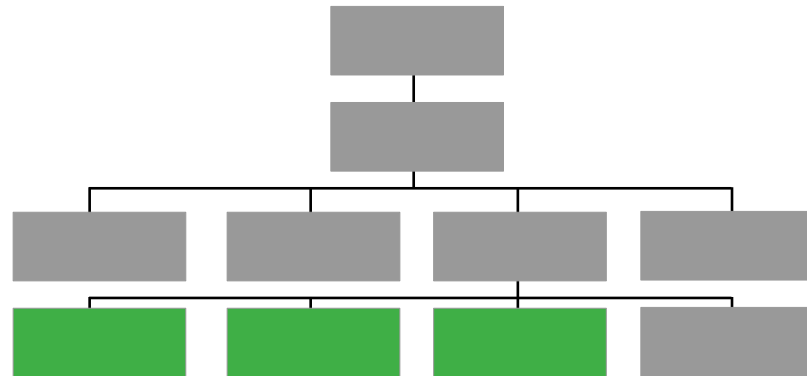
Usa o valor do atributo id de um elemento (que deve ser único dentro da página)

- **`querySelector()`** : Usa um seletor CSS e retorna o primeiro elemento correspondente



Acessando os Elementos

- Acessando múltiplos elementos (Lista de Nós).



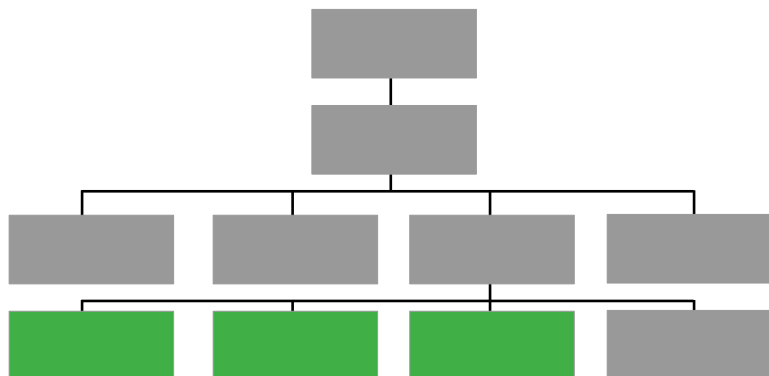
- **`getElementsByName()`** :
método retorna uma coleção de elementos com um atributo 'nome' especificado

- **`getElementsByTagName()`** :
método retorna uma coleção de elementos com uma tag específica.



Acessando os Elementos

- Acessando múltiplos elementos (Lista de Nós).



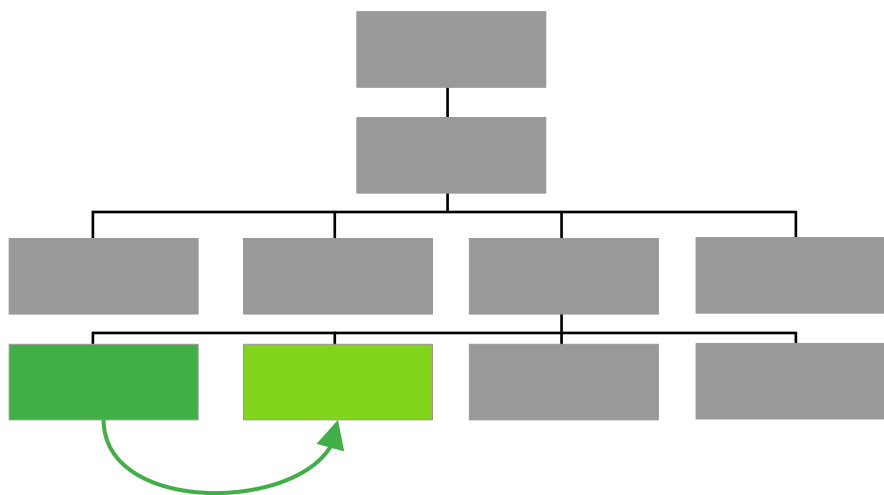
- **`getElementsByClassName()`** : método retorna coleção de elementos que possuem um valor class atributo

- **`querySelectorAll()`** : Usa um seletor CSS para selecionar todos os elementos correspondentes



Acessando os Elementos

- Atravessando entre os nós dos elementos: mover de um nó para um nó mais próximo ou relacionado



- **previousSibling/nextSibling**: Seleciona o irmão anterior ou próximo da árvore DOM.

- **parentNode**: Seleciona o pai do nó do elemento atual (que retornará apenas um elemento)

- **firstChild/lastChild**: Seleciona o primeiro ou último nó filho do elemento atual.



Métodos para acesso individual

- `getElementById()` é o mais rápido e eficiente caminho para acessar um elemento. O ideal que não haja mais de um elemento com mesmo atributo ID.

Refere-se ao objeto 'Document'

O **método** indica qual elemento deve ser encontrado baseado no valor do seu **atributo ID**.

`document.getElementById('one')`

Operador

Indica que o **método** à direita está sendo **aplicado** no **nó** da esquerda

Parâmetro

Valor do atributo **ID** do elemento correspondente



getElementById('id')

HTML

```
<h1 id="header">Lista de Compras</h1>
<h2>Comprar Mantimentos</h2>
<ul>
  <li id="one" class="hot"><em>Picanha</em> fresca</li>
  <li id="two" class="hot">Castanha</li>
  <li id="three" class="hot">Mel</li>
  <li id="four">Vinagre</li>
</ul>
```

JS

```
//Seleciona o elemento e armazena em uma variável
let el = document.getElementById('one');

//Altera o valor do atributo Classe.
el.className = 'cool'
```

CSS

```
.cool{
  background-color: blue;
  color: white;
}
```



getElementById('id')

Antes

Lista de Compras

Comprar Mantimentos

- Picanha fresca
- Castanha
- Mel
- Vinagre

</h1>

```
"><em>Picanha</em> fresca</li>
">Castanha</li>
">Mel</li>
">
```

HTML

```
//Altera o valor do atributo C
el.className = 'cool'
```

```
.cool{
  background-color: blue;
  color: white;
}
```

Depois

Lista de Compras

Comprar Mantimentos

- Picanha fresca
- Castanha
- Mel
- Vinagre

Métodos para acesso individual

- `querySelector()` é um método recente adicionado ao DOM. É flexível por causa do parâmetro ser um seletor CSS. Este método retorna apenas o primeiro elemento da lista de nós.

Refere-se ao objeto 'Document'

O **método** indica qual elemento deve ser encontrado baseado no valor do seu **Seletor CSS**.

`document.querySelector('li.hot')`

Operador

Indica que o **método** à direita está sendo **aplicado** no **nó**

Parâmetro

Valor do Seletor CSS do elemento correspondente



querySelector('Seletor CSS')

HTML

```
<h1 id="header">Lista de Compras</h1>
<h2>Comprar Mantimentos</h2>
<ul>
  <li id="one" class="hot"><em>Picanha</em> fresca</li>
  <li id="two" class="hot">Castanha</li>
  <li id="three" class="hot">Mel</li>
  <li id="four">Vinagre</li>
</ul>
```

JS

```
//Seleciona o elemento e armazena em uma variável
let el = document.querySelector('li.hot');

//Altera o valor do background e cor da Font.
el.style.background = 'green';
el.style.color='black';
```



querySelector('Seletor CSS')

HTML

```
<h1 id="header">Lista de Compras</h1>
<h2>Comprar Mantimentos</h2>
<ul>
  <li id="one" class="hot"><em>Picanha</em> fresca</li>
  <li id="two" class="hot">Castanha</li>
  <li id="three" class="hot">Doce de leite</li>
  <li id="four" class="hot">Doce de leite</li>
</ul>
```

Propriedade Style sobrescreve o valor CSS do elemento identificado

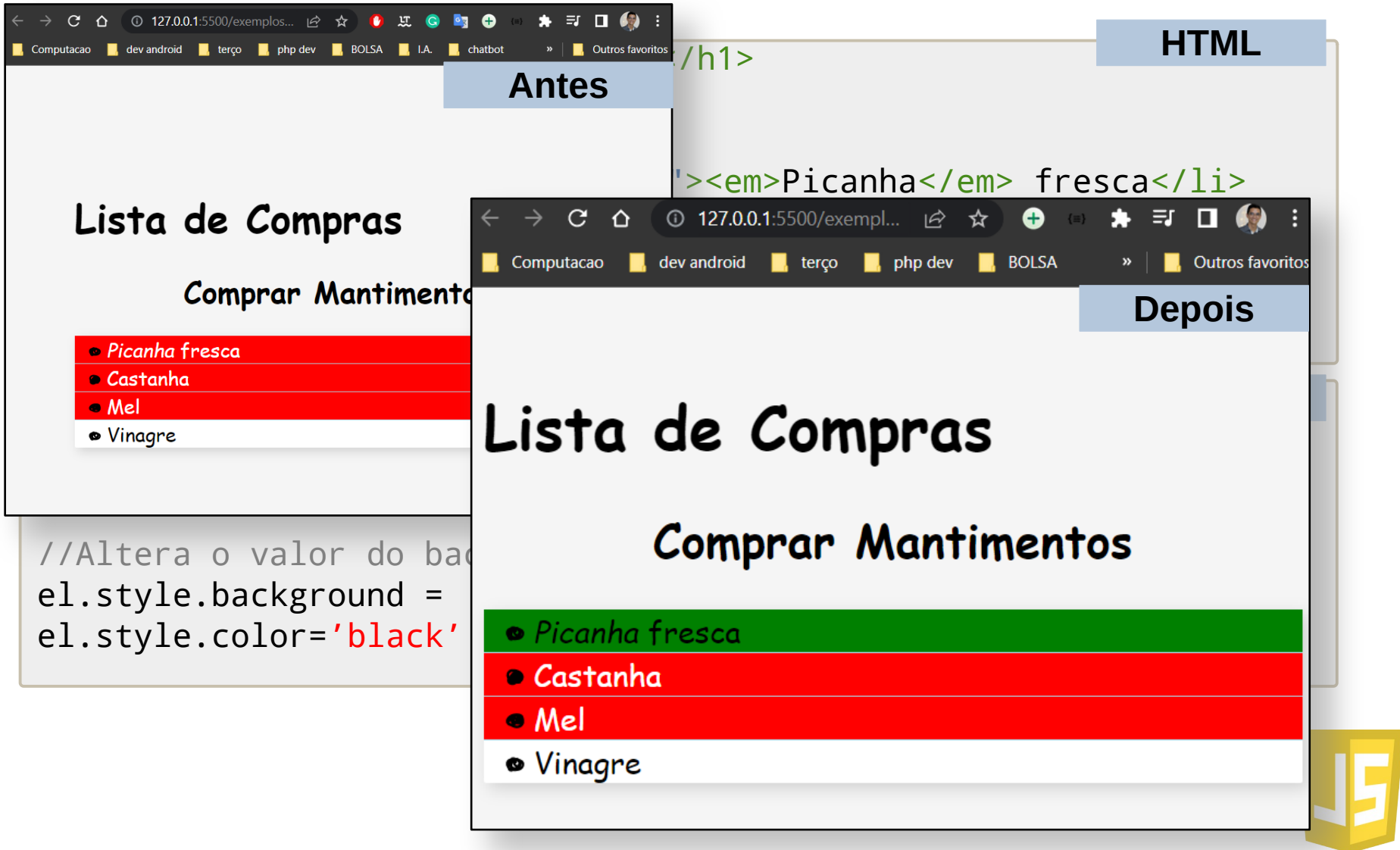
JS

```
//Seleciona o elemento pelo seletor CSS e atribui o valor
let el = document.querySelector('li.hot');

//Altera o valor do background e cor da Font.
el.style.background = 'green';
el.style.color='black';
```



querySelector('Seletor CSS')



Propriedade Style JavaScript

- Propriedades CSS e suas equivalências na linguagem JavaScript:

CSS

- `background-color`
- `border-radius`
- `font-size`
- `list-style-type`
- `word-spacing`
- `z-index`

JavaScript

- `backgroundColor`
- `borderRadius`
- `fontSize`
- `listStyleType`
- `wordSpacing`
- `zIndex`

Lista completa em:

<https://www.javascripttutorial.net/javascript-dom/javascript-style/>



Propriedade Style JavaScript

- Outras formas de mudar os valores e propriedades CSS.
- inline

JS

```
//Selecione o elemento e armazene em uma variável
let el = document.querySelector('li');

//Altera o valor do background e cor da Font.
el.style.cssText = 'color:red;background-color:yellow'
```

- Método `setAttribute()`

JS

```
//Selecione o elemento e armazene em uma variável
let el = document.querySelector('li');

//Altera o valor do background e cor da Font.
el.style.setAttribute('style', 'color:red;background-color:yellow')
```



Propriedade Style JavaScript

- `getComputedStyle()` é um método retorna um objeto que contém o estilo computado de um elemento. Seus parâmetros são:
- **Element**: é o elemento que você deseja que retorne os estilos computados
- **PseudoElement**: especifica o pseudo-elemento a ser correspondido. O padrão é null

```
//Seleciona o elemento e armazena em uma variável
let link = document.querySelector('a');
let style = getComputedStyle(link, ':hover');

console.log(style) //resultado é a cor observada na ação :hover
```



Propriedade Style JavaScript

- Qual valor irá retornar do `getComputedStyle()` ?

HTML

```
<p class="message" style="color:red" >  
  Isto é um JS getComputedStyle() demo!  
</p>
```

CSS

```
.message{  
  background-color: #fff3d4;  
  border: 1px solid #f6b73c;  
  padding: 20px;  
  color: black;  
}
```

JS

```
//Seleciona o elemento e armazena em uma variável  
let message = document.querySelector('.message');  
let style = getComputedStyle(message);  
console.log('color: ', style.color);  
console.log('background color: ', style.backgroundColor);
```



StyleSheets JavaScript

- Também podemos usar as propriedades **setProperty** e **removeProperty** para manipular os estilos

HTML

```
<div id="icomp">Instituto de Computação</div>
<button id="add">Adicionar Estilos</button>
<button id="clean">Limpar Estilo</button>
```

JS

```
let icomp = document.getElementById("icomp");
document.getElementById("add").onclick = function() {
    icomp.style.setProperty("background-color", "lightgray");
    icomp.style.setProperty("color", "red");
    icomp.style.setProperty("font-size", "32px");
}
document.getElementById("clean").onclick = function() {
    icomp.style.removeProperty("background-color");
    icomp.style.removeProperty("color");
    icomp.style.removeProperty("font-size");
}
```



StyleSheets JavaScript

usar as propriedades **setProperty** e **removeProperty** para manipular os estilos

HTML

```
Instituto de Computação</div>
Adicionar Estilos</button>
```

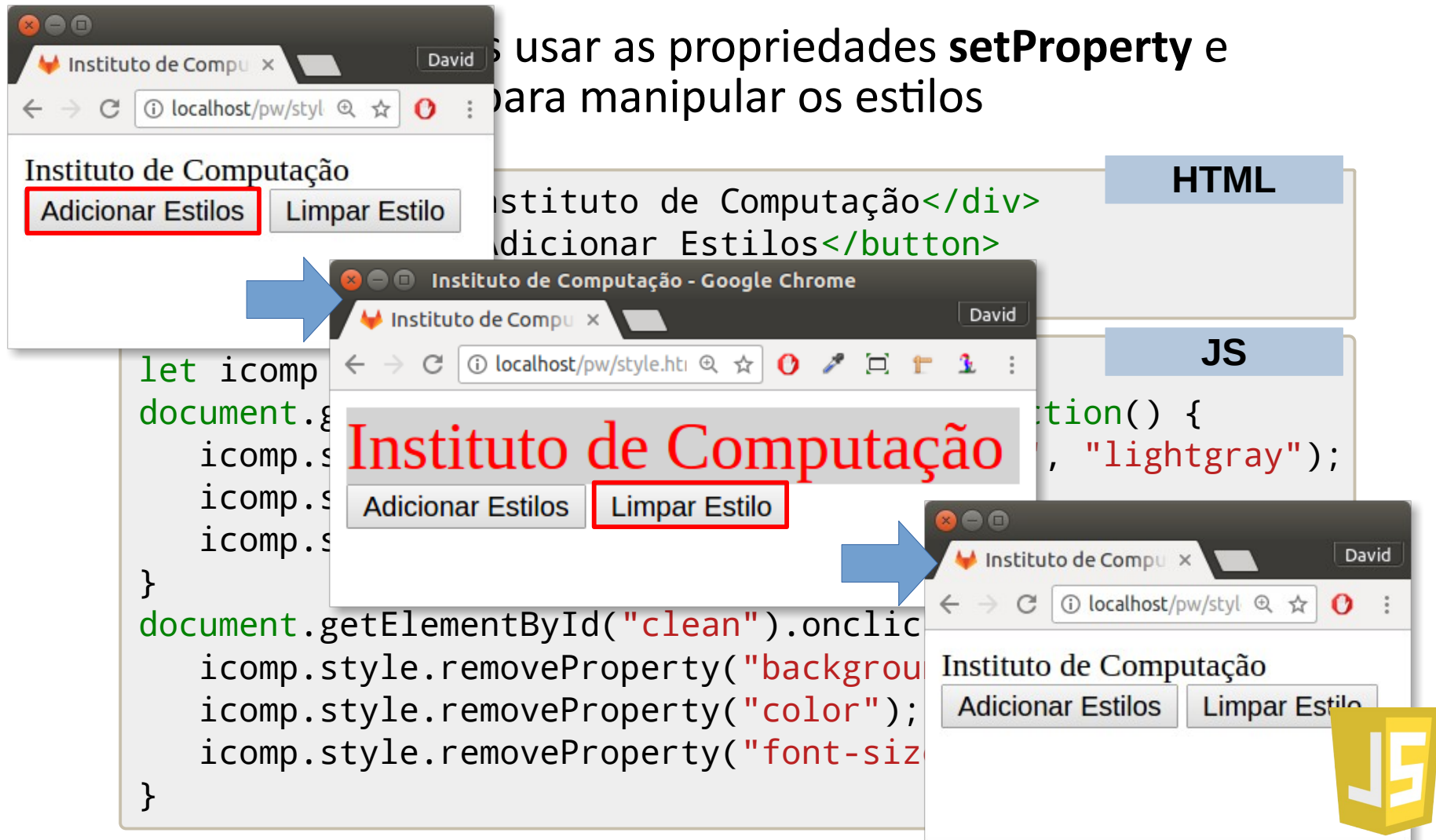
JS

```
let icomp
document.g
icomp.s
icomp.s
icomp.s
}
document.getElementById("clean").onclick
icomp.style.removeProperty("backgrou
icomp.style.removeProperty("color");
icomp.style.removeProperty("font-siz
}
```

Browser State 1: Instituto de Computação, Adicionar Estilos (highlighted), Limpar Estilo

Browser State 2: Instituto de Computação - Google Chrome, Instituto de Computação (highlighted), Adicionar Estilos, Limpar Estilo (highlighted)

Browser State 3: Instituto de Computação, Adicionar Estilos, Limpar Estilo



StyleSheets JavaScript

- **Atividade:** Dado o seguinte HTML, crie um JS que modifique o estilo do background <p> através do código a partir do prompt.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset=utf-8 />
    <title>JS DOM parágrafo CSS</title>
  </head>
  <body>
    <p id='text'>Exercício Javascript Exercicios</p>
    <div>
      <button id="addstyle" onclick= "addStyle()">Style</button>
      <button id="clearstyle" onclick= "removeStyle()">Remove</button>
    </div>
  </body>
</html>
```

HTML

```
#text{
  font-size: 14pt;
  font-family: "Comic Sans MS";
  color: "white";
}
```

CSS



Manipulação dos atributos dos Elementos

- O processo de atualização dos atributos dos elementos é realizado em duas etapas:
 - 1) Selecionar o elemento nó que irá aplicar a atualização do atributo
 - 2) Aplicar um dos métodos ou propriedades abaixo.

Método	Descrição
<code>getAttribute()</code>	Obtém o valor de um atributo
<code>hasAttribute()</code>	Verifica se um elemento nó possui um atributo específico
<code>setAttribute()</code>	Define o valor de um atributo
<code>removeAttribute()</code>	Remove um atributo do elemento nó
Propriedade	Descrição
<code>className</code>	obtém ou define um valor para o atributo <i>class</i>
<code>id</code>	obtém ou define um valor para o atributo <i>id</i>
<code>classList</code>	retorna uma coleção ativa de classes CSS



Propriedade className

- `className` é a propriedade de um elemento que retorna uma lista separada por espaços de classes CSS do elemento como uma string

HTML

```
<ul id="menu" class="vertical main">
  <li> Home </li>
  <li> Serviços </li>
  <li> Sobre </li>
  <li> Contato </li>
</ul>
```

JS

```
//Seleciona o elemento e armazena em uma variável
let menu = document.querySelector('#menu');

console.log(menu.className); //output: vertical main
```



Propriedade className

- Concatenando Classes

HTML

```
<ul id="menu" class="vertical main">
  <li> Home </li>
  <li> Serviços </li>
  <li> Sobre </li>
  <li> Contato </li>
</ul>
```

JS

```
//Seleciona o elemento e armazena em uma variável
let menu = document.querySelector('#menu');
menu.className += 'new';

console.log(menu.className); //output: vertical main new
```



Propriedade classList

- `classList` é a propriedade que retorna a coleção ativa de classes CSS do elemento.

```
//Seleciona o elemento e armazena em uma variável
let el = document.querySelector('#menu');
const classes = el.classList;

for(let cssClass of classes)
    console.log(cssClass); //output: vertical main
```

- Os métodos `add()` e `remove()` respectivamente são utilizadas para adicionar e remover classes CSS a partir de uma lista de classes de um elemento.
- `replace()` pode ser utilizado para trocar uma classe existente por uma nova
- `contains()` é um método que verifica se um elemento de uma lista de classes contém uma classe específica.



Propriedade classList

- Os métodos `add()` e `remove()` respectivamente são utilizadas para adicionar e remover classes CSS a partir de uma lista de classes de um elemento.

```
//Seleciona o elemento e armazena em uma variável
let el = document.querySelector('#menu');
el.classList.add('new');

console.log(el.className); //output: vertical main new
```

```
//Seleciona o elemento e armazena em uma variável
let el = document.querySelector('#menu');
el.classList.remove('vertical');

console.log(el.className); //output: main new
```



Propriedade classList

- **replace()** pode ser utilizado para trocar uma classe existente por uma nova.

```
//Seleciona o elemento e armazena em uma variável  
let el = document.querySelector('#menu');  
el.classList.replace('main', 'principal');  
  
console.log(classes); //output: vertical principal
```

- **contains()** é um método que verifica se um elemento de uma lista de classes contém uma classe específica.

```
//Seleciona o elemento e armazena em uma variável  
let el = document.querySelector('#menu');  
el.classList.contains('principal'); //return true
```



Método para atualizar atributo

Utilize os métodos vistos até agora para encontrar o elemento nó da árvore DOM

O **método** obtém o valor do atributo que está referenciado no parâmetro.

```
document.getElementById('one').getAttribute('class')
```

Operador

Indica que o subsequente **método** será **aplicado** no **nó específico à esquerda**



Verificação de atributo

- O método `hasAttribute()` é utilizado para verificar se um elemento nó possui algum atributo especificado. O nome do atributo é passado por parâmetro. O método retorna *true* ou *false*.

JS

```
//Seleciona o elemento e armazena em uma variável
const el = document.querySelector('li');

if (el.hasAttribute('class')) {
  let attr = el.getAttribute('class'); //output: hot
  alert("Yes attribute exist!" + attr);
}
else {
  alert("No attribute doesn't exist!");
}
```



Verificação de atributo

- O método `hasAttribute()` é utilizado para verificar se um **elemento** **nó** possui algum atributo específico. O atributo a ser verificado é passado por parâmetro. O método retorna `true` se o atributo existe, caso contrário retorna `false`.

Método verifica
Se o atributo
Existe no elemento
nó.

JS

```
//Seleciona o elemento e o atributo em uma variável
const el = document.querySelector('li');

if (el.hasAttribute('class')) {
  let attr = el.getAttribute('class'); //output: hot
  alert("Yes attribute exist!" + attr);
}
else {
  alert("No attribute doesn't exist!")
}
```

Se verdadeiro o seu
Valor é capturado
Pelo método
`getAttribute()`



Criando Atributos e alterando seu valor

- A propriedade `className` permite modificar o valor de um atributo `class`. Caso o atributo não exista, a propriedade irá criar e inserir o valor especificado.

JS

```
//Seleciona o elemento e armazena em uma variável
let el = document.getElementById('one');

//altera o atributo class
el.className = 'thunder'

//Seleciona o quarto elemento e armazena em uma variável
let el2 = document.getElementsByTagName('li').item(3);

//adiciona um atributo para ele
el2.setAttribute('class', 'thunder');
```



Criando Atributos e alterando seu valor

- A propriedade `className` permite modificar o valor de um atributo `class`. Caso o atributo não exista, a propriedade irá criar e inserir o valor especificado.

JS

```
//Seleciona o elemento e armazena em uma variável
let el = document.getElementById('thunder');

//altera o atributo class
el.className = 'thunder';

//Seleciona o quarto elemento
let el2 = document.getElementsByTagName('p')[3];

//adiciona um atributo para ele
el2.setAttribute('class', 'thunder');
```

O método `setAttribute()` permite atualizar o valor de qualquer atributo. Precisando passar dois parâmetros: 'nome atributo' e valor para o atributo.



Criando Atributos e alterando seu valor

- ✓ Às vezes é melhor atualizar os atributos utilizando propriedades (**className** ou **id**).
- ✓ Contudo, os métodos `getAttribute()` e `setAttribute()` permite que você defina qualquer atributo que desejar nos nós.
- ✗ Cuidado quando atualizar um atributo, principalmente *class*, pois o seu valor pode estar associado algum tipo de seletor CSS, e portanto pode alterar a aparência dos elementos.



Removendo Atributos

- O método `removeAttribute()` permite remover atributos de um elemento nó, apenas passando o nome como parâmetro.

JS

```
//Seleciona o elemento e armazena em uma variável
let el = document.getElementById('one');

//Verifica se o atributo class existe
if (el.hasAttribute('class')) {
    //Se existir remove o atributo do elemento nó
    el.removeAttribute('class');
}
```



Removendo Atributos

- O método `removeAttribute()` permite remover atributos de um elemento nó, apenas passando o nome como parâmetro.

```
//Seleciona o elemento
let el = document.querySelector('div')

//Verifica se o atributo class existe
if (el.hasAttribute('class')) {
  //Se existir remove o atributo do elemento nó
  el.removeAttribute('class');
}
```

Tentar remover algum atributo inexistente não causará nenhum erro, mas uma boa prática, é verificar antes de remover.



Métodos para acesso Lista de Nós

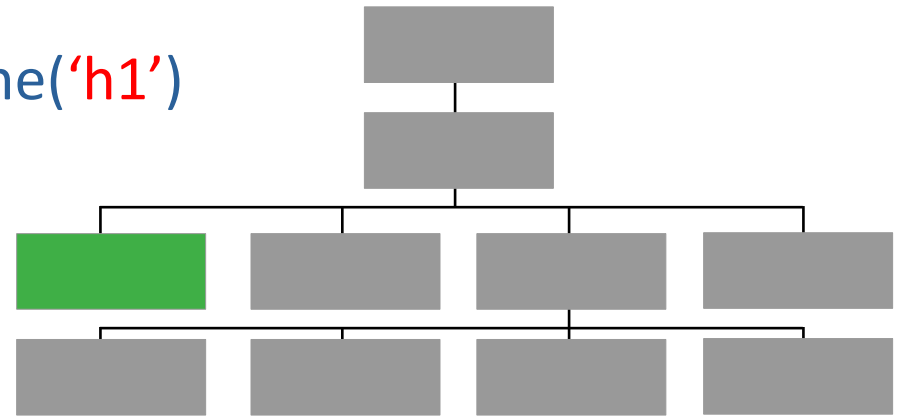
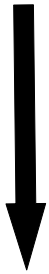
- Uma Lista de nó são uma coleção de elementos. Cada nó possui um número index.
- As listas de nós parecem arrays e são numeradas como arrays, mas na verdade não são arrays; eles são um tipo de objeto chamado coleção
- A ordem dos nós são armazenadas conforme a disposição na árvore DOM.
- Propriedade **length** e **item()** são utilizados para acessar os valores dos elementos.



getElement^sByTagName('tag')

- `getElementsByTagName()` o método retorna uma lista de nós devido ao potencial de retornar mais de um elemento

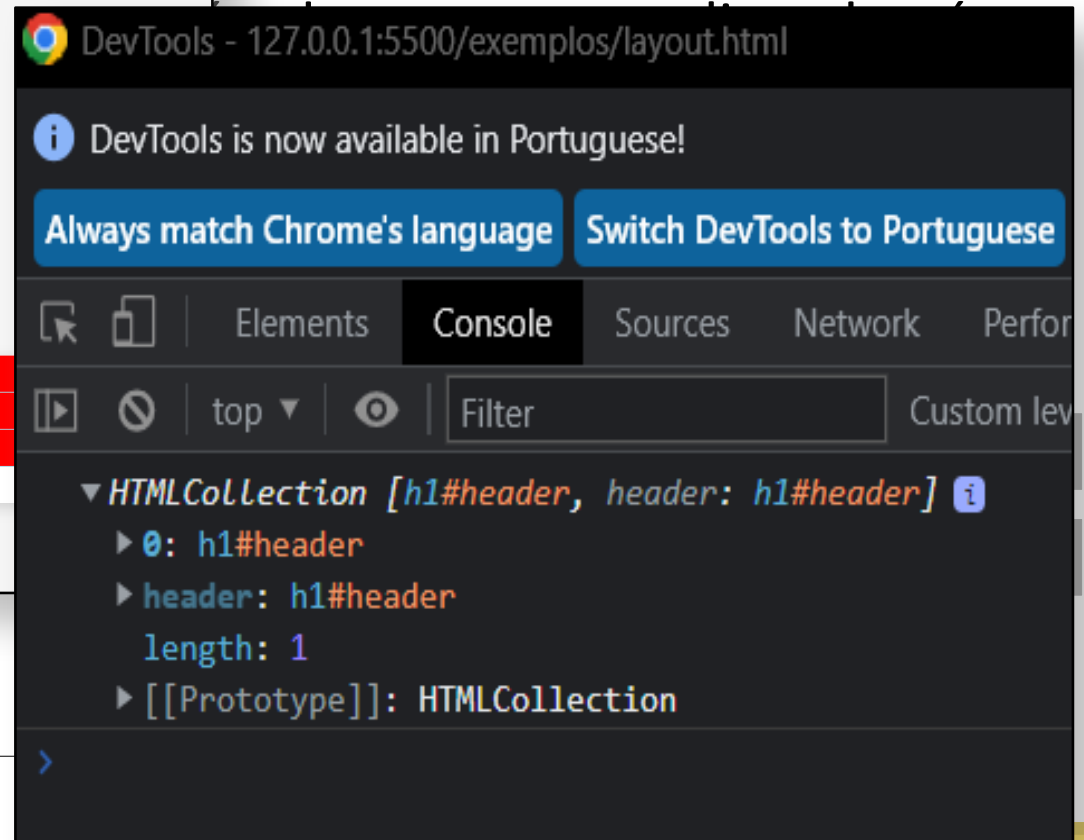
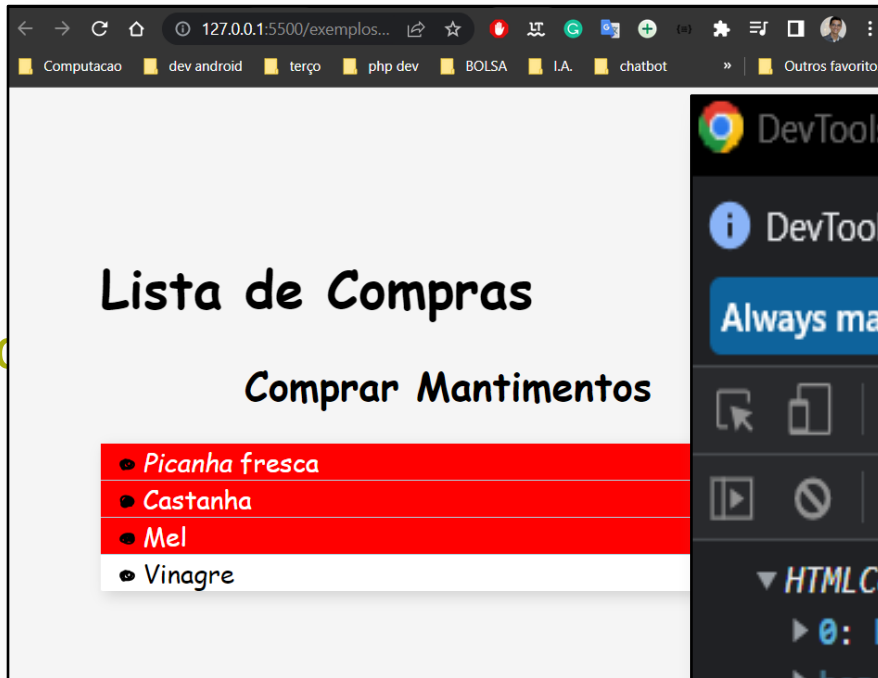
`document.getElementsByTagName('h1')`



Index	Elemento
0	<h1>



getElement^sByTagName('tag')



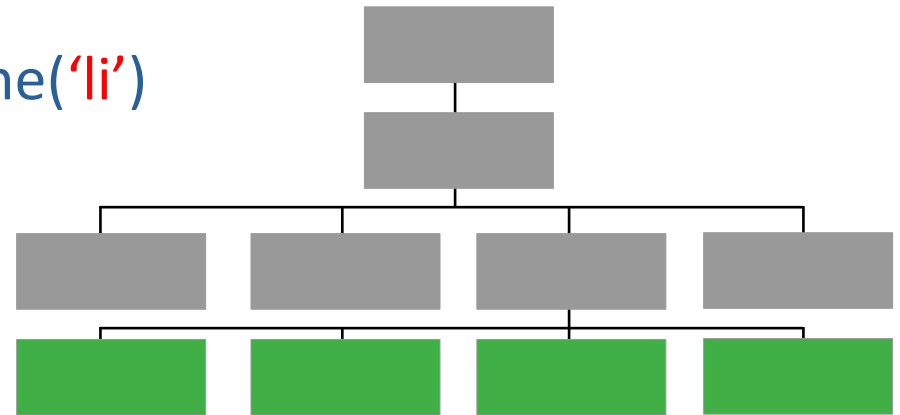
Index	Elemento
0	<h1>



getElement^sByTagName('tag')

- `getElementsByTagName()` o método retorna uma lista de nós devido ao potencial de retornar mais de um elemento

`document.getElementsByTagName('li')`



Index	Elemento
0	<code><li id="one" class="hot"></code>
1	<code><li id="two" class="hot"></code>
2	<code><li id="three" class="hot"></code>
3	<code><li id="four"></code>



getElementsByTagName('tag')

todo retorna uma lista de nós
mais de um elemento

Lista de Compras

Comprar Mantimentos

- Picanha fresca
- Castanha
- Mel
- Vinagre

Index

Elemento

0

<li id="one" class="hot">

1

<li id="two" class="hot">

2

<li id="three" class="hot">

3

<li id="four" class="hot">

DevTools is now available in Portuguese!

Always match Chrome's language

Switch DevTools to Portuguese

Don't show again



Elements

Console



top



Filter

Custom levels

1 Issue: 1

list.js:4

HTMLCollection(4) [li#one.hot, li#two.hot, li#three.hot, li#four, one: li#one.hot, two: li#two.hot, three: li#three.hot, four: li#four]

▶ 0: li#one.hot

▶ 1: li#two.hot

▶ 2: li#three.hot

▶ 3: li#four

▶ four: li#four

▶ one: li#one.hot

▶ three: li#three.hot

▶ two: li#two.hot

length: 4

▶ [[Prototype]]: HTMLCollection



Atributos DOM

- **Atividade:** Os métodos `getAttribute()` e `setAttribute()` permite pesquisar e/ou definir qualquer atributo que desejar para os elementos nós. Utilizando o HTML abaixo, crie um JS que altere a visualização do conteúdo do elemento na DOM 'secret'. Teste as opções 'display:none', 'visible:hidden', `classList` 'remove()'

HTML

```
<p data-classified = 'secret'> A senha do cofre é 123987654.</p>
<p data-classified = 'unclassified'> John Wick 4 é top!<p>
<p data-classified = 'secret'> Saldo no banco é de R$ 0,0002.<p>
```

JS

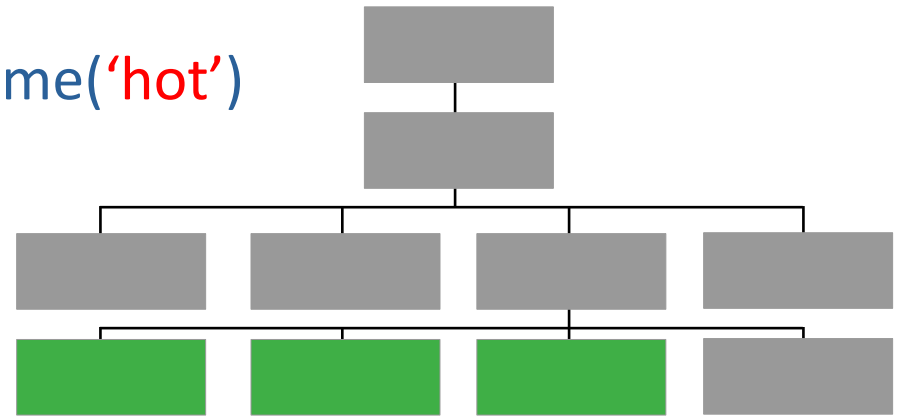
```
<script>
  // Insira a solução aqui
</script>
```



getElement^sByClassName('class')

- `getElementsByClassName()` o método retorna uma lista de nós devido ao potencial de retornar mais de um elemento

`document.getElementsByClassName('hot')`



Index	Elemento
0	<code><li id="one" class="hot"></code>
1	<code><li id="two" class="hot"></code>
2	<code><li id="three" class="hot"></code>



getElementByClassName('class')

- todo retorna uma lista de nós

do

Lista de Compras

Comprar Mantimentos

- Picanha fresca
- Castanha
- Mel
- Vinagre

Index

Elemento

0

<li id="one" class

1

<li id="two" clas

2

<li id="three" cla

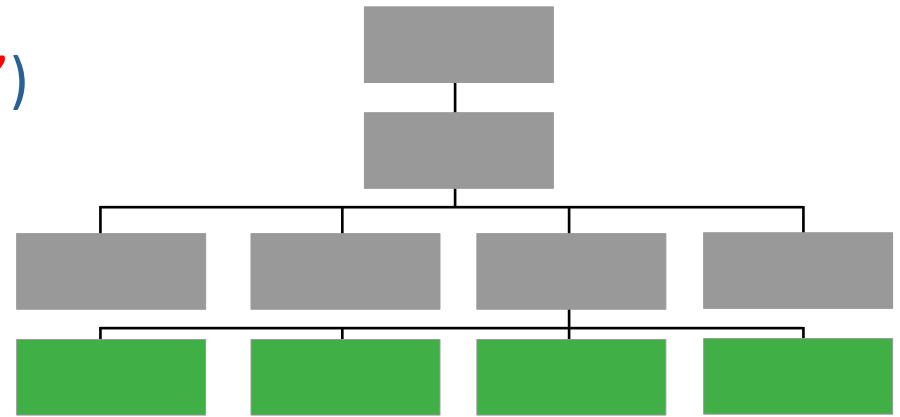
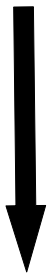
```
DevTools - 127.0.0.1:5500/exemplos/layout...
DevTools is now available in Portuguese!
Always match Chrome's language
Switch DevTools to Portuguese Don't show again
Elements Console >>
top Filter 1 hidden
Custom levels 1 Issue: 1
list.js:4
HTMLCollection(3) [li#one.hot, li#two.hot,
▼ li#three.hot, one: li#one.hot, two: li#two.hot, thr
ee: li#three.hot] ⓘ
  ▶ 0: li#one.hot
  ▶ 1: li#two.hot
  ▶ 2: li#three.hot
  ▶ one: li#one.hot
  ▶ three: li#three.hot
  ▶ two: li#two.hot
  length: 3
  ▶ [[Prototype]]: HTMLCollection
>
```



querySelectorAll('seletor CSS')

- `querySelectorAll()` o método retorna uma lista de nós devido ao potencial de retornar mais de um elemento

`document.querySelectorAll('li[id]')`

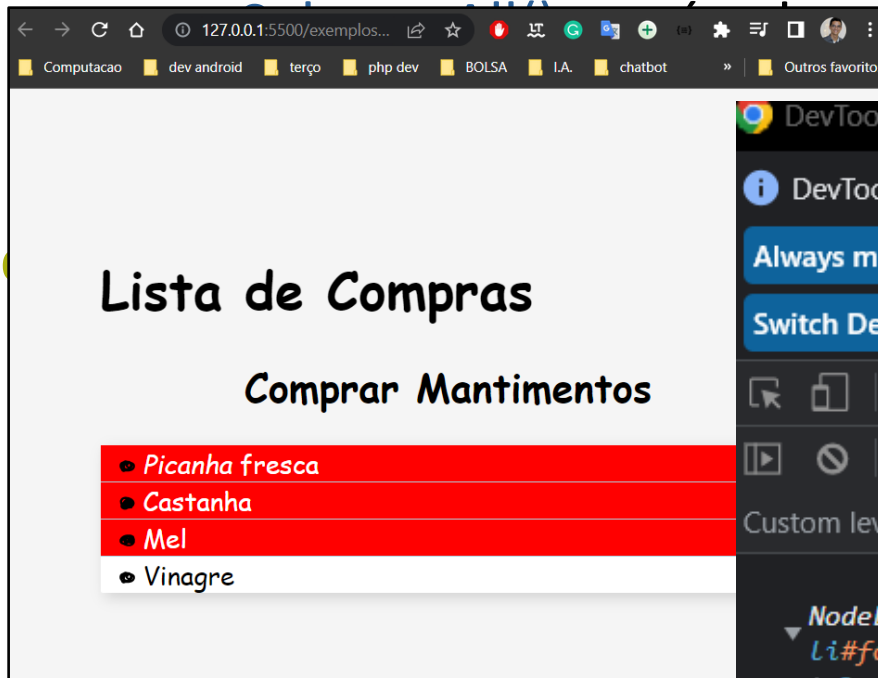


Index	Elemento
0	<code><li id="one" class="hot"></code>
1	<code><li id="two" class="hot"></code>
2	<code><li id="three" class="hot"></code>
3	<code><li id="four"></code>



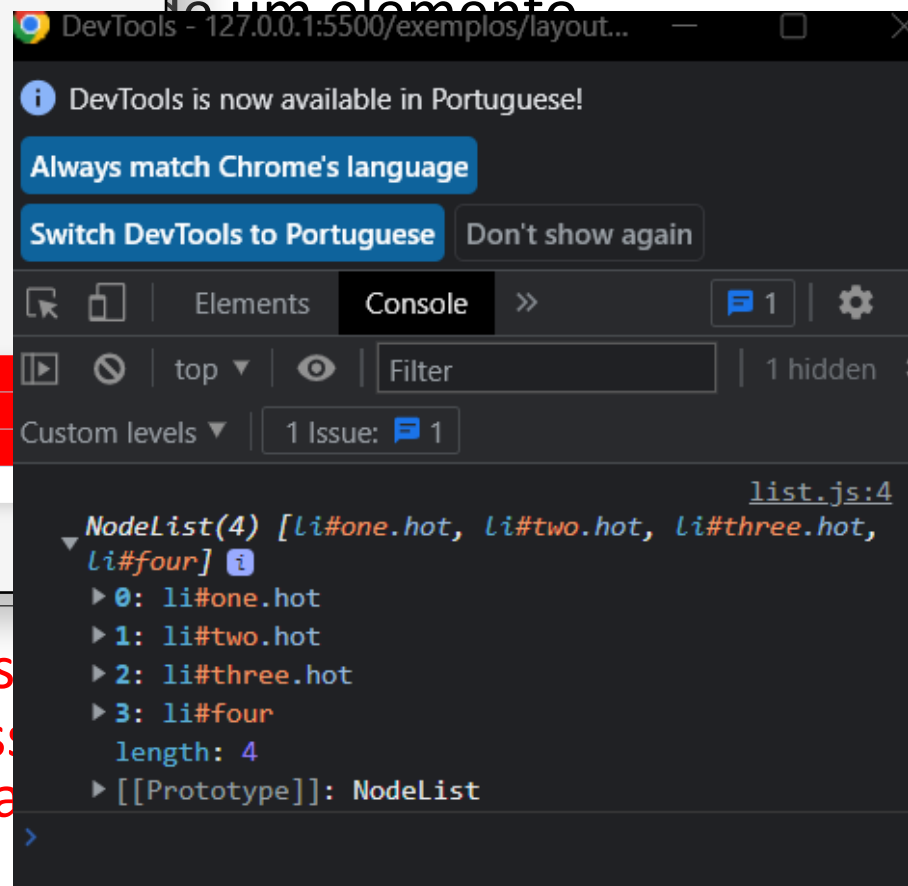
querySelectorAll('seletor CSS')

torna uma lista de nós devido
a um elemento



0
1
2
3

<li id="one" class
<li id="two" class
<li id="three" cla
<li id="four" >



Selecionando um elemento a partir de uma Lista de nós

- Existem dois meios para selecionar um elemento a partir de uma lista de nó: método `item()` e sintaxe array.
- Método `item()` → retorna um nó individual a partir da lista de nós

```
let el = document.getElementsByClassName( 'hot' )

if(el.length >= 1){
    let firstItem = el.item(0);
}
```



Selecionando um elemento a partir de uma Lista de nós

- Existem dois meios para selecionar um elemento a partir de uma lista de nós: método `item()`
- Método `item()` → retorna um nó

Seleciona os elementos que possuem
O atributo `class='hot'` e são
armazenados em uma variável

```
let el = document.getElementsByClassName( 'hot' )

if(el.length >= 1){
    let firstItem = el.item(0);
}
```



Selecionando um elemento a partir de uma Lista de nós

- Existem dois meios para selecionar um elemento a partir de uma lista de nó: método `item()` e sintaxe array.
- Método `item()` → retorna um nó individual a partir da lista de nós

Usando a propriedade *length* para checar se existe elementos na coleção, então a declaração `if` é executada

```
let el = document.querySelector('hot')

if(el.length >= 1){
  let firstItem = el.item(0);
}
```



Selecionando um elemento a partir de uma Lista de nós

- Existem dois meios para selecionar um elemento a partir de uma lista de nó: método `item()` e sintaxe array.
- Para utilizar o método `item()` é necessário Passar o valor do index como parâmetros Para poder acessar o elemento.** individual a partir da lista de

```
let el = document.getElementsByClassName( 'hot' )

if(el.length >= 1){
    let firstItem = el.item(0);
}

console.log(firstItem)
```



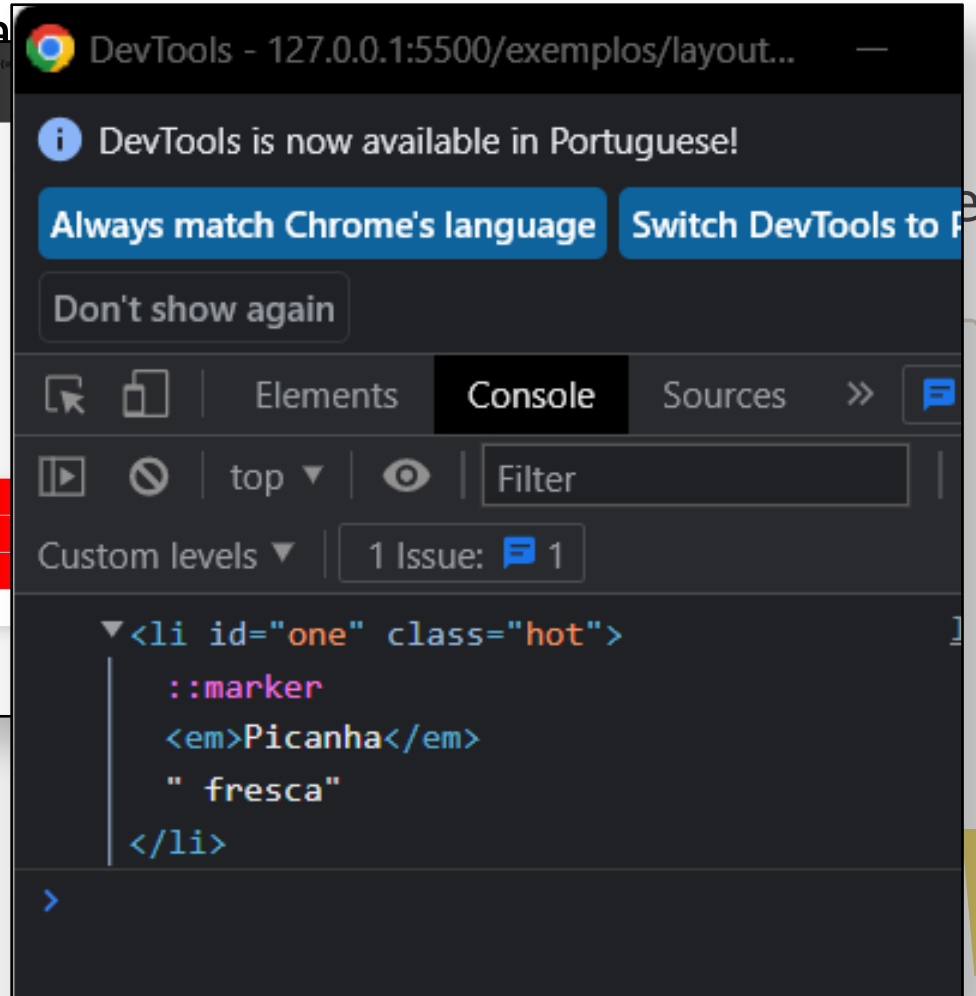
Selecionando um elemento a partir de uma Lista de nós

- Existem dois meios para

Lista de Compras

Comprar Mantimentos

- Picanha fresca
- Castanha
- Mel
- Vinagre



Selecionando um elemento a partir de uma Lista de nós

- Existem dois meios para selecionar um elemento a partir de uma lista de nó: método `item()` e sintaxe array.
- Método **sintaxe array** → permite utilizar colchetes para acessar um elemento.

```
let el = document.getElementsByClassName( 'hot' )

if(el.length >= 1){
    let firstItem = el[0];
}
```

A sintaxe array é preferida porque é mais prática Em relação ao método `item()`



Utilizando Loop em uma Lista de nós

- **Atividade:** Com base no Slide anterior, aplique a [sintaxe array](#) ou método [item\(\)](#) para aplicar a classe 'cool' no seguinte HTML. Utilize o método [querySelectorAll\(\)](#)

HTML

```
<div id="page">
  <h1 id="header">Lista de Compras</h1>
  <h2>Comprar Mantimentos</h2>
  <ul>
    <li id="one" class="hot"><em>Picanha</em> fresca</li>
    <li id="two" class="hot">Castanha</li>
    <li id="three" class="hot">Mel</li>
    <li id="four">Vinagre</li>
  </ul>
  <script src="js/list.js"></script>
</div>
```

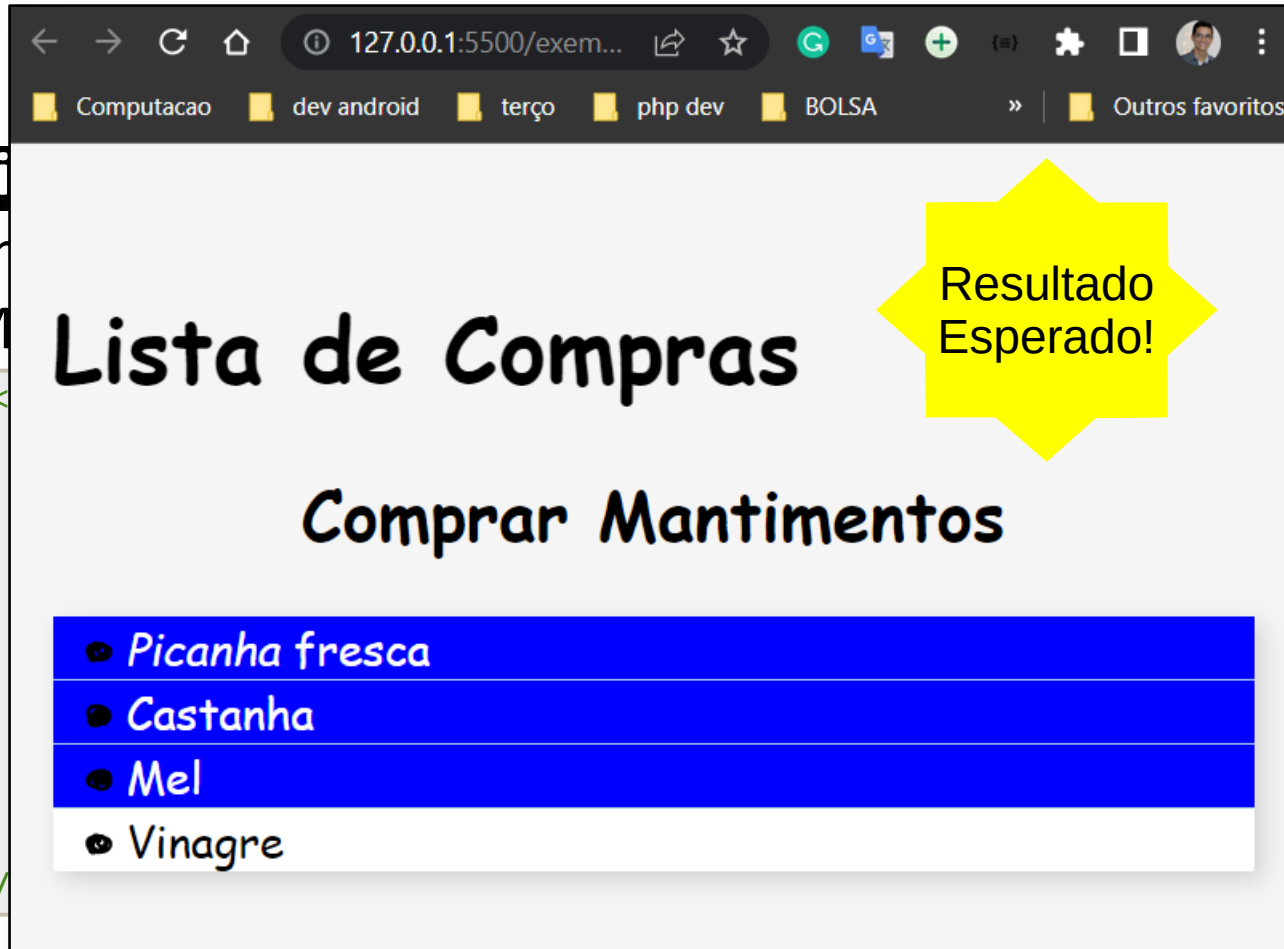
CSS

```
.cool{
  background-color: blue;
  color: white;
}
```



Utilizando Loop em uma Lista de nós

- **Ativi**
ou m
HTM



xe array

te

ML

/li>

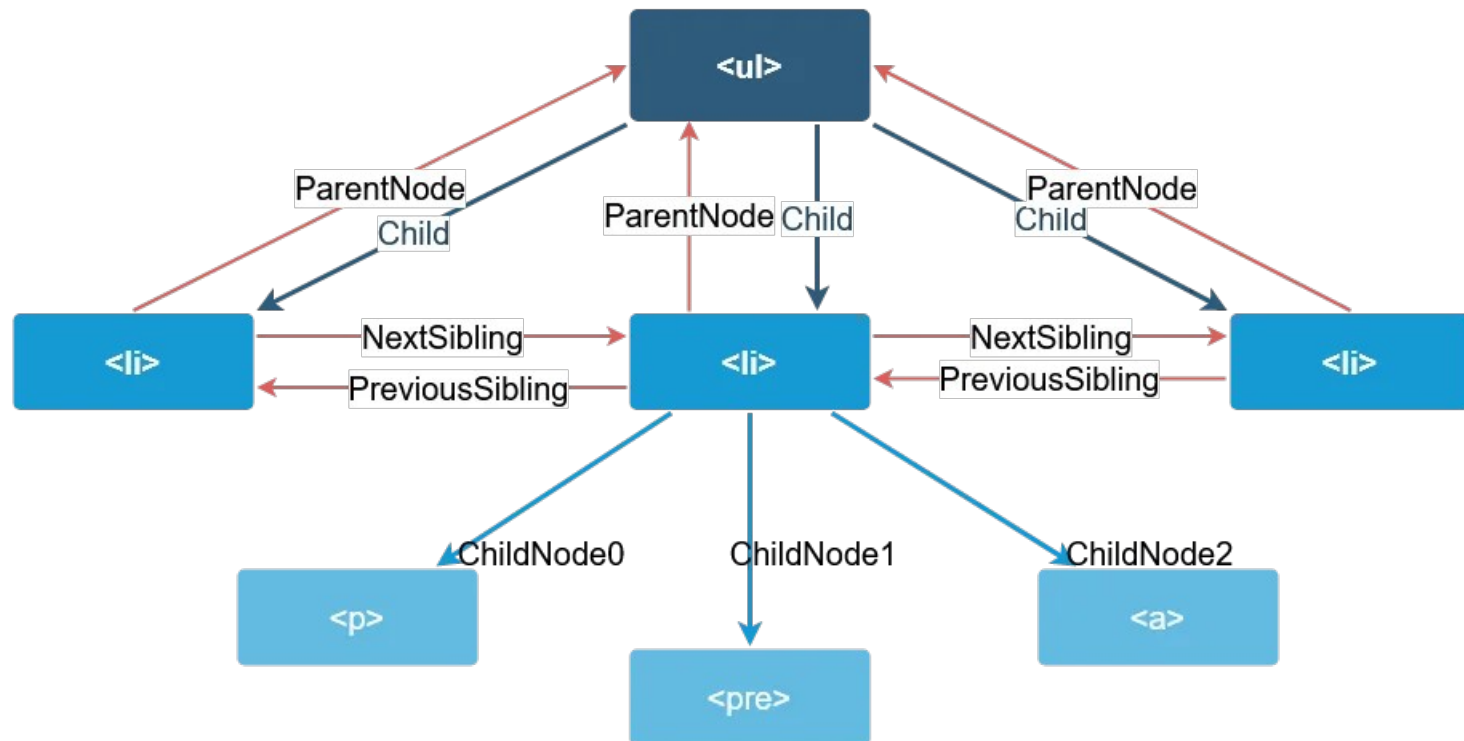
CSS

```
.cool{  
  background-color: blue;  
  color: white;  
}
```




Atravessando a DOM

- Atravessar (caminhada ou navegação) na DOM é o ato de selecionar nós na árvore a partir de outros nós.
- Podemos **mover para baixo**, **para cima** ou **para os lados** na árvore DOM.



Atravessando a DOM

- Mover para baixo na DOM

- 
- **firstChild** → retorna o primeiro filho do elemento
 - **firstElementChild** → Retorna o primeiro elemento filho do pai
 - **lastChild** → Retorna o último filho do elemento
 - **lastElementChild** → Retorna o último elemento filho do pai
 - **childNodes** → Retorna todos os filhos do elemento como uma coleção de array
 - **children** → Retorna todos os filhos que são elemento como uma coleção de array



Atravessando a DOM

- `firstChild` & `lastChild`

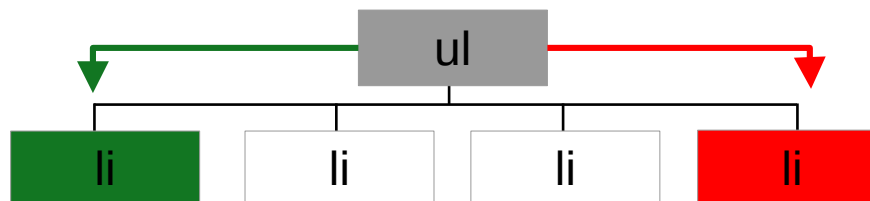
HTML

```
<ul>
  <li id="one"    class="hot"><em>Picanha</em> fresca</li>
  <li id="two"    class="hot">Castanha</li>
  <li id="three"  class="hot">Mel</li>
  <li id="four">Vinagre</li>
</ul>
```

JS

```
//Seleciona o nó filho
let startItem = document.getElementsByTagName('ul')[0];
let firstItem = startItem.firstChild;
let lastItem = startItem.lastChild;

firstItem.style.backgroundColor = 'green';
lastItem.style.backgroundColor = 'red';
```



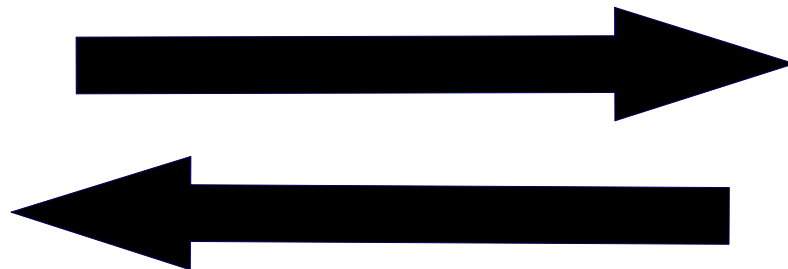
Atravessando a DOM

- Mover para cima na DOM
 - `parentNode` → retorna o nó pai do elemento
 - `parentElement` → Retorna o nó do elemento pai do elemento



Atravessando a DOM

- **Mover para os lados na DOM**
 - **nextSibling** → Retorna o **nó irmão** que é o próximo filho de seu pai
 - **nextElementSibling** → Retorna o **elemento irmão** que é o próximo filho de seu pai
 - **previousSibling** → Retorna o **nó irmão** que é um filho anterior de seu pai
 - **previousElementSibling** → Retorna o **elemento irmão** que é um filho anterior de seu pai



Atravessando a DOM

- **previousSibling** & **nextSibling**

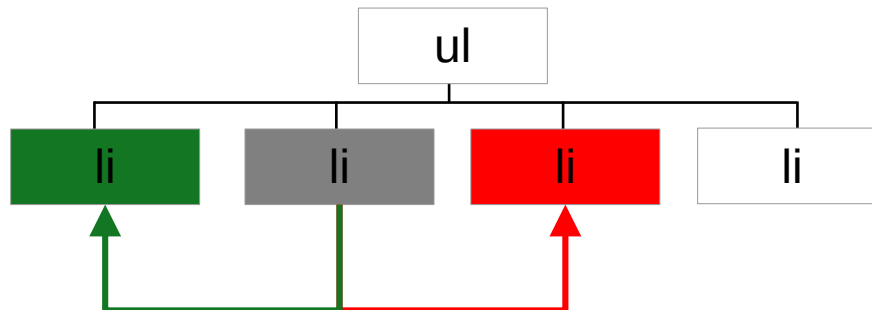
HTML

```
<ul>
  <li id="one"    class="hot"><em>Picanha</em> fresca</li>
  <li id="two"    class="hot">Castanha</li>
  <li id="three"  class="hot">Mel</li>
  <li id="four">Vinagre</li>
</ul>
```

JS

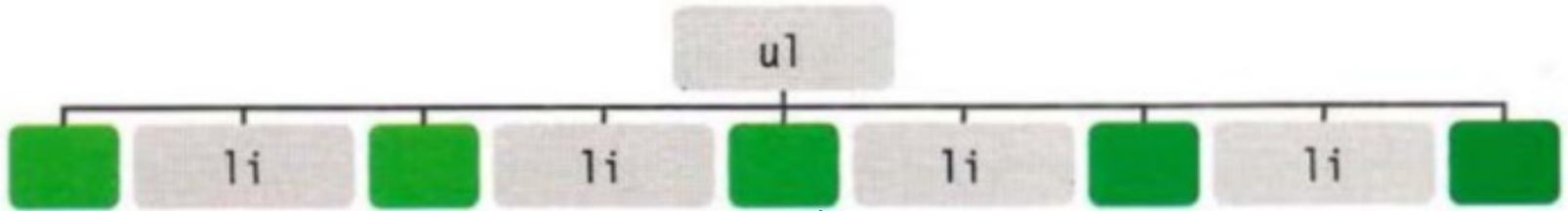
```
//Seleciona o nó filho
let startItem = document.getElementById('two');
let prevItem = startItem.previousSibling;
let nextItem = startItem.nextSibling;

prevItem.style.backgroundColor = 'green';
nextItem.style.backgroundColor = 'red';
```



ALERTA | Espaços em branco na DOM

- A maioria dos navegadores, trata os espaços em branco entre os elementos como um nó de texto



Os espaços estão
Caracterizados na
Cor verde



ALERTA Espaços em branco na DOM

HTML

```
<div id="parent">
  <h1>This is heading</h1>
  <p>This is paragraph</p>
</div>
```

JS

```
<script>
  const element = document.getElementById("parent");
  const first = element.firstChild;
  console.log(first); // text node
  console.log(first.nodeName); // output: #text
</script>
```



ALERTA Espaços em branco na DOM

```
<div id="parent">  
  <h1>This is heading</h1>  
  <p>This is paragraph</p>  
</div>
```

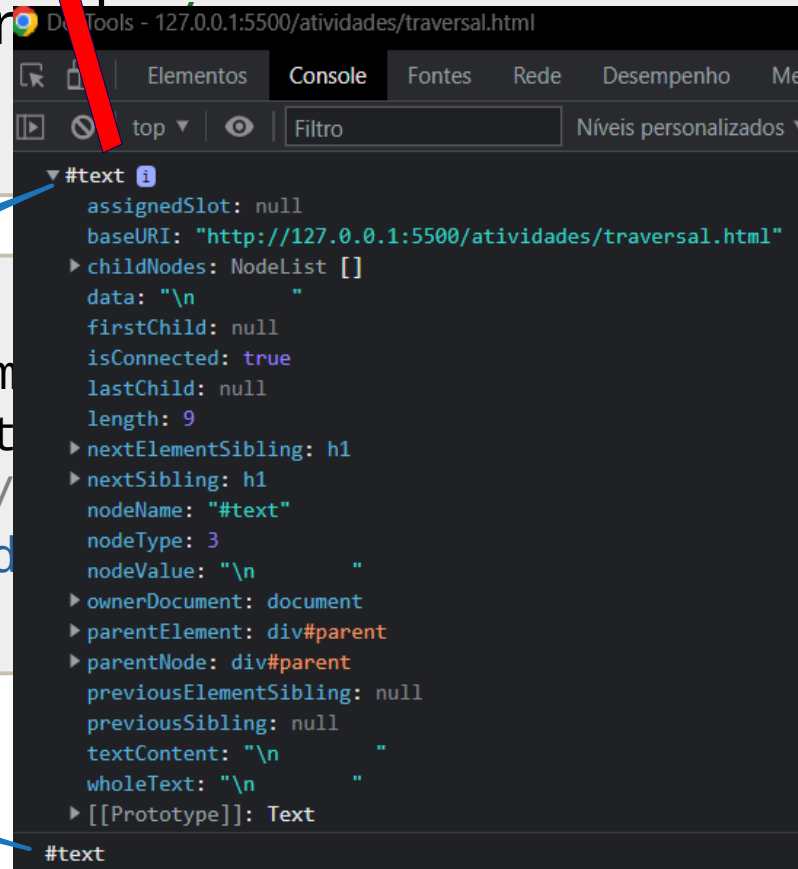
Localização
do Nó #text

HTML

```
<script>  
  const first = document.querySelector('h1');  
  const firstNode = first.firstChild;  
  console.log(first);  
  console.log(firstNode);  
</script>
```

Nó

nodeName



ALERTA Espaços em branco na DOM

- Duas SOLUÇÕES possíveis

Eliminar os espaços
no HTML

HTML

```
<div id="parent"><h1>This is heading</h1><p>
This is paragraph</p></div>
```

Mudar a propriedade

JS

```
<script>
  const element = document.getElementById("parent");
  const first = element.firstElementChild;
  console.log(first); // output: <h1>This is heading</h1>
  console.log(first.nodeName); // output: H1
</script>
```



ALERTA Espaços em branco na DOM

- Duas SOLUÇÕES possíveis

Eliminar os espaços

Mudar a p

HTML

```
<div id="p">
  This is p
```

```
ing</h1><p>
```

```
<h1>This is heading</h1>
```

H1

```
>
```

JS

```
<script>
  const ele
  const fir
  console.log
  console.log(first.nodeName); // output: H1
</script>
```

```
d("parent");
d;
heading</h1>
```



Atravessando a DOM

- **Atividade:** Com base nas propriedades de navegação na árvore DOM, a partir do Código abaixo, informe: o Elemento anterior e o próximo do Elemento id= 'two' e o Primeiro e último filho da div = 'page'

HTML

```
<div id="page">
  <h1 id="header">Lista de Compras</h1>
  <h2>Comprar Mantimentos</h2>
  <ul>
    <li id="one" class="hot"><em>Picanha</em> fresca</li>
    <li id="two" class="hot">Castanha</li>
    <li id="three" class="hot">Mel</li>
    <li id="four">Vinagre</li>
  </ul>
  <script src="js/list.js"></script>
</div>
```

CSS

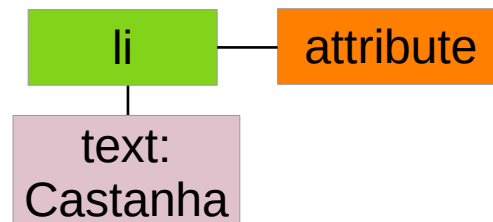
```
.cool{
  background-color: blue;
  color: white;
}
```



Modificando o conteúdo (texto) dos elementos da DOM

- Para trabalhar com o conteúdo dos elementos, você pode:
 - 1) **Navegue até os nós de texto.** Isso funciona melhor quando o elemento contém apenas texto, nenhum outro elemento
 - 2) **Trabalhe com o elemento recipiente.** Isso permite que você acesse seus nós de texto e elementos filho. Funciona melhor quando um elemento tem nós de texto e elementos filhos que são irmãos

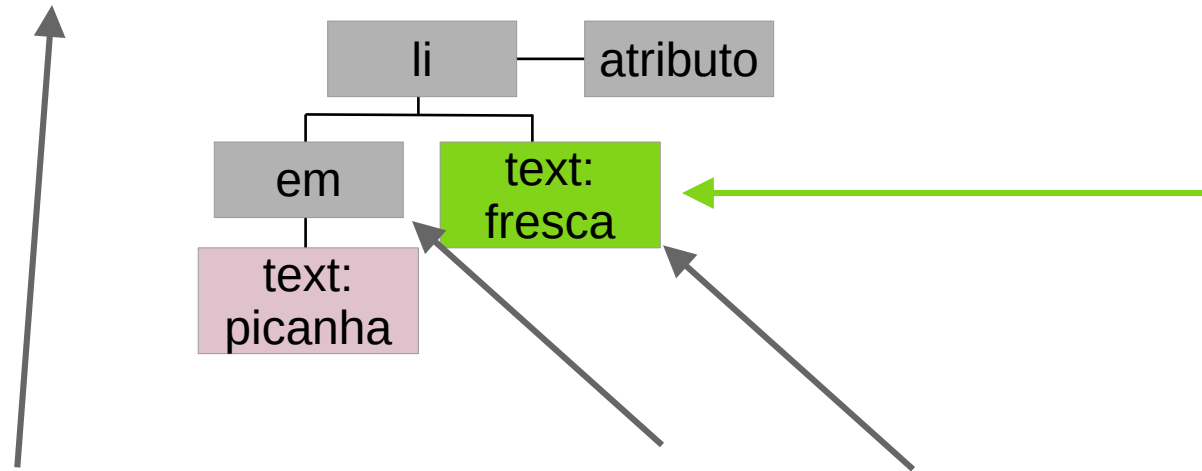
```
<li id="two" class="hot">Castanha</li>
```



Modificando o conteúdo (texto) dos elementos da DOM

- Ao selecionar um nó de texto, você pode recuperar ou corrigir o conteúdo dele usando a propriedade **nodeValue**.

```
<li id="one"><em>Picanha</em> fresca</li>
```



```
let text =  
document.getElementById('one').firstChild.nextSibling.nodeValue;
```

```
console.log(text); //output: fresca
```

JS



Modificando o conteúdo (texto) dos elementos da DOM

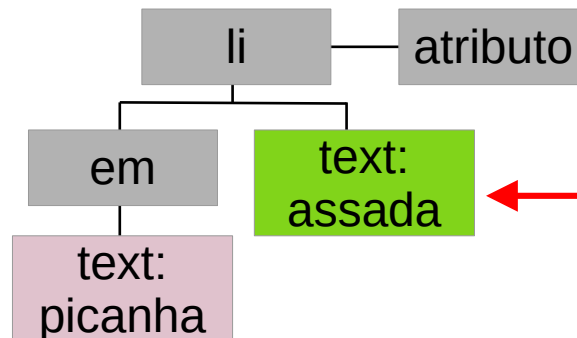
- Ao selecionar um nó de texto, você pode recuperar ou corrigir o conteúdo dele usando a propriedade **nodeValue**.

```
let text = document.getElementById('one')  
  
text.firstChild.nextSibling.nodeValue = ' assada';
```

JS

- Resultado.**

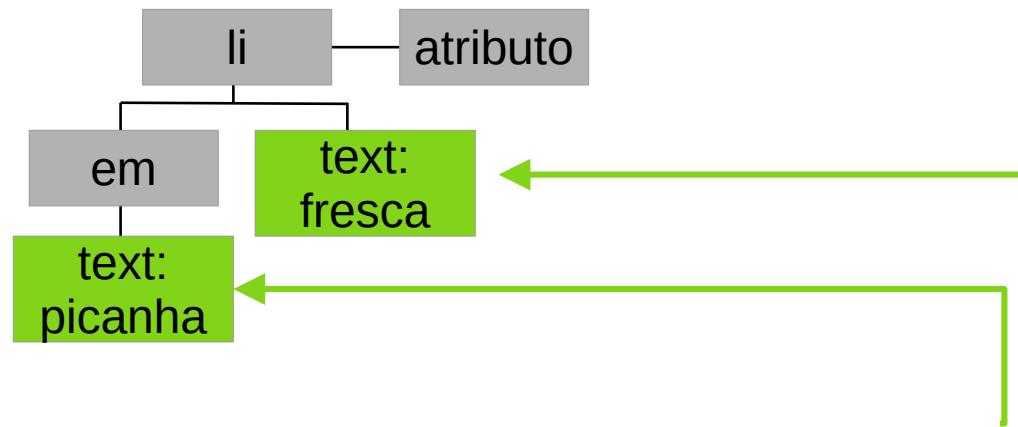
`<li id="one">Picanha assada`



Modificando o conteúdo (texto) dos elementos da DOM

- A propriedade **textContent** permite coletar ou atualizar apenas o texto que está no elemento recipiente

```
<li id="one"><em>Picanha</em> fresca</li>
```



```
let text = document.getElementById('one').textContent;
```

```
console.log(text); //output: Picanha fresca
```

JS



Modificando o conteúdo (texto) dos elementos da DOM

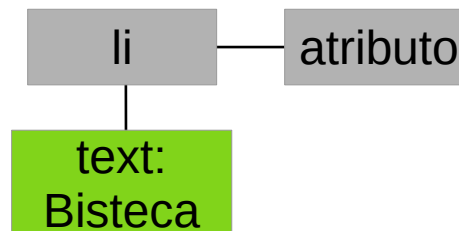
- A propriedade **textContent** permite coletar ou atualizar apenas o texto que está no elemento recipiente

```
let text = document.getElementById('one')  
  
text.textContent = 'Bisteca';
```

JS

- Resultado.**

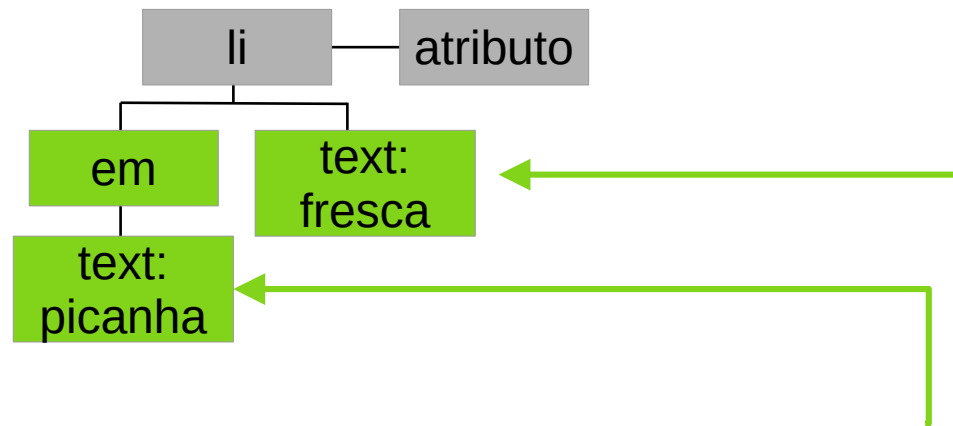
`<li id="one">Bisteca`



Modificando o conteúdo (texto) dos elementos da DOM

- Usando a propriedade **innerHTML**, você pode acessar e corrigir o conteúdo de um elemento, incluindo qualquer elemento filho.

```
<li id="one"><em>Picanha</em> fresca</li>
```



```
let text = document.getElementById('one').innerHTML;
```

```
console.log(text); //output: <em>Picanha</em> fresca
```

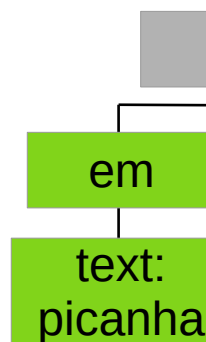
JS



Modificando o conteúdo (texto) dos elementos da DOM

- Usando a propriedade **innerHTML**, você pode acessar e corrigir o conteúdo de um elemento, incluindo qualquer elemento filho.

```
<li id="one"><em>Picanha</em> fresca</li>
```



a propriedade **innerHTML** obterá o conteúdo de um elemento e o retornará como uma string longa, incluindo qualquer marcação que o elemento contenha.

```
let text = document.getElementById('one').innerHTML;
```

```
console.log(text); //output: <em>Picanha</em> fresca
```

JS



Modificando o conteúdo (texto) dos elementos da DOM

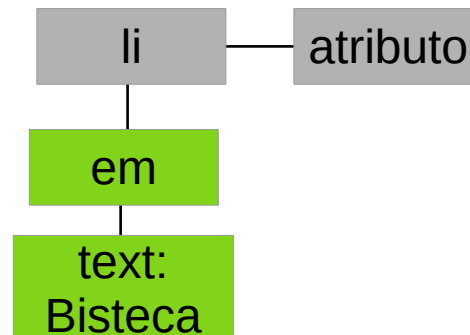
- A propriedade **textContent** permite coletar ou atualizar apenas o texto que está no elemento recipiente

```
let text = document.getElementById('one')  
  
text.innerHTML = '<em>Bisteca</em>';
```

JS

- Resultado.**

```
<li id="one"><em>Bisteca</em></li>
```



Modificando o conteúdo (texto) dos elementos da DOM

Método	Descrição
nodeValue	Acessar elemento texto a partir do nó
textContent	Verificar/Adicionar apenas texto
innerText	Verificar/Adicionar apenas texto
innerHTML	Verificar/Adicionar texto e HTML



Modificando o conteúdo (texto) dos elementos da DOM

- **Vantagens** em utilizar a propriedade **innerHTML**
 - Poder usá-lo para adicionar novas marcações (tags) usando menos código do que os métodos de manipulação de DOM
 - Pode ser mais rápido do que a manipulação de DOM ao adicionar muitos novos elementos a uma página da web.
 - É uma maneira simples de remover todo o conteúdo de um elemento (atribuindo-lhe uma string em branco)



Modificando o conteúdo (texto) dos elementos da DOM

- **Desvantagens** em utilizar a propriedade **innerHTML**
 - Ele não deve ser usado para adicionar conteúdo proveniente de um usuário (como um nome de usuário ou comentário de blog), pois pode representar um risco de segurança significativo.
 - Pode ser difícil isolar elementos únicos que você deseja atualizar em um fragmento DOM maior.
 - Os manipuladores de eventos podem não funcionar conforme o esperado



Referências

- DUCKETT, Jon. **JavaScript & JQuery: Interactive Front-End Web Development**. Wiley. 2010.
- Digital Ocean. Disponível em:
<https://www.digitalocean.com/community/tutorial_series/understanding-the-dom-document-object-model>. Acessado em 15/04/2023
- OSBORN, Jeremy and SMITH, Jennifer. **Web Design with HTML and CSS Digital Classroom**. Wiley. 2011
- W3Schools. Disponível em <https://www.w3schools.com/jsref/dom_obj_document.asp>
Acessado em 15/04/2023.