



JavaScript

Introdução ao DOM

Prof.: MSc. Arcanjo Miguel Mota Lopes

Manipulando árvore DOM

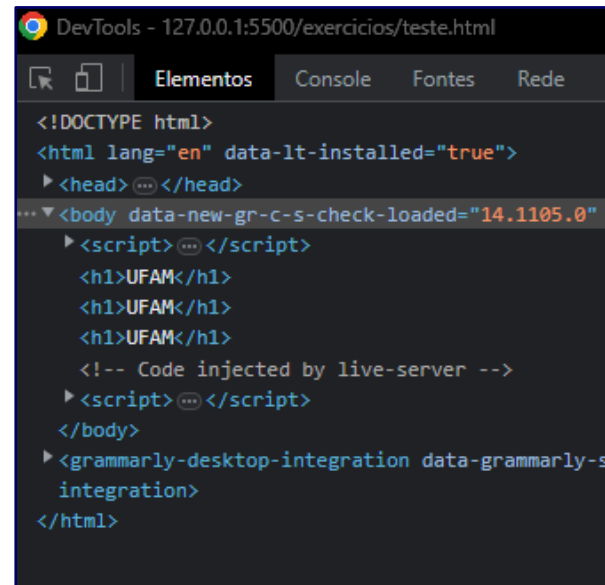
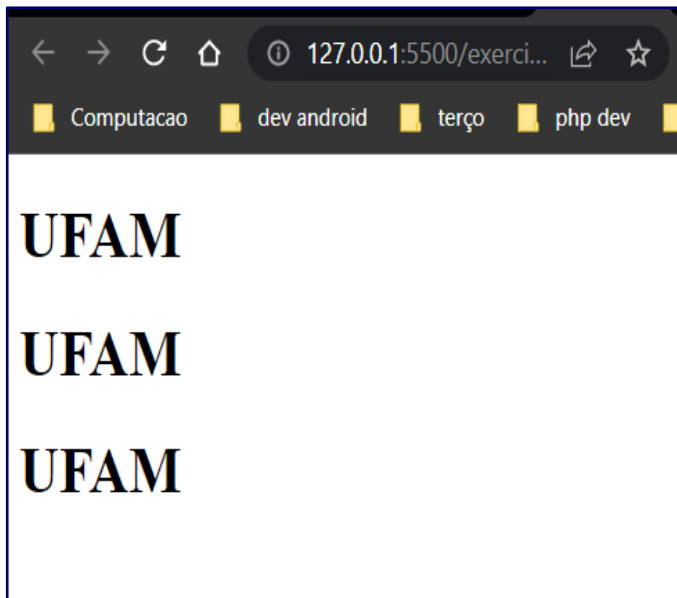
- Nos últimos slides, foi apresentado como usar a DOM para manipular elementos individuais das páginas
 - Foi mostrado, por exemplo, como mudar os atributos e os estilos dos elementos
- Conforme já vimos, isso é possível porque existe um link em tempo real entre a DOM e o documento HTML
- É possível usar esse link para irmos além das manipulações apresentadas, por exemplo, mudando a estrutura do documento HTML



Manipulando árvore DOM

- **document.write**

```
document.write("<h1>UFAM</h1>");  
document.write("<h1>UFAM</h1>");  
document.write("<h1>UFAM</h1>");
```

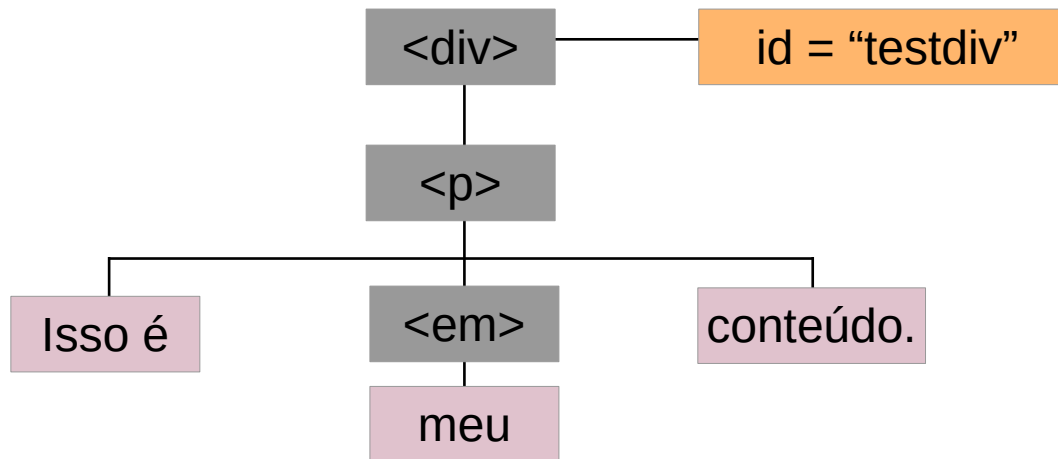


Manipulando árvore DOM

- innerHTML

```
<div id="testdiv">  
  <p> Isso é <em>meu</em> conteúdo. </p>  
</div>
```

HTML



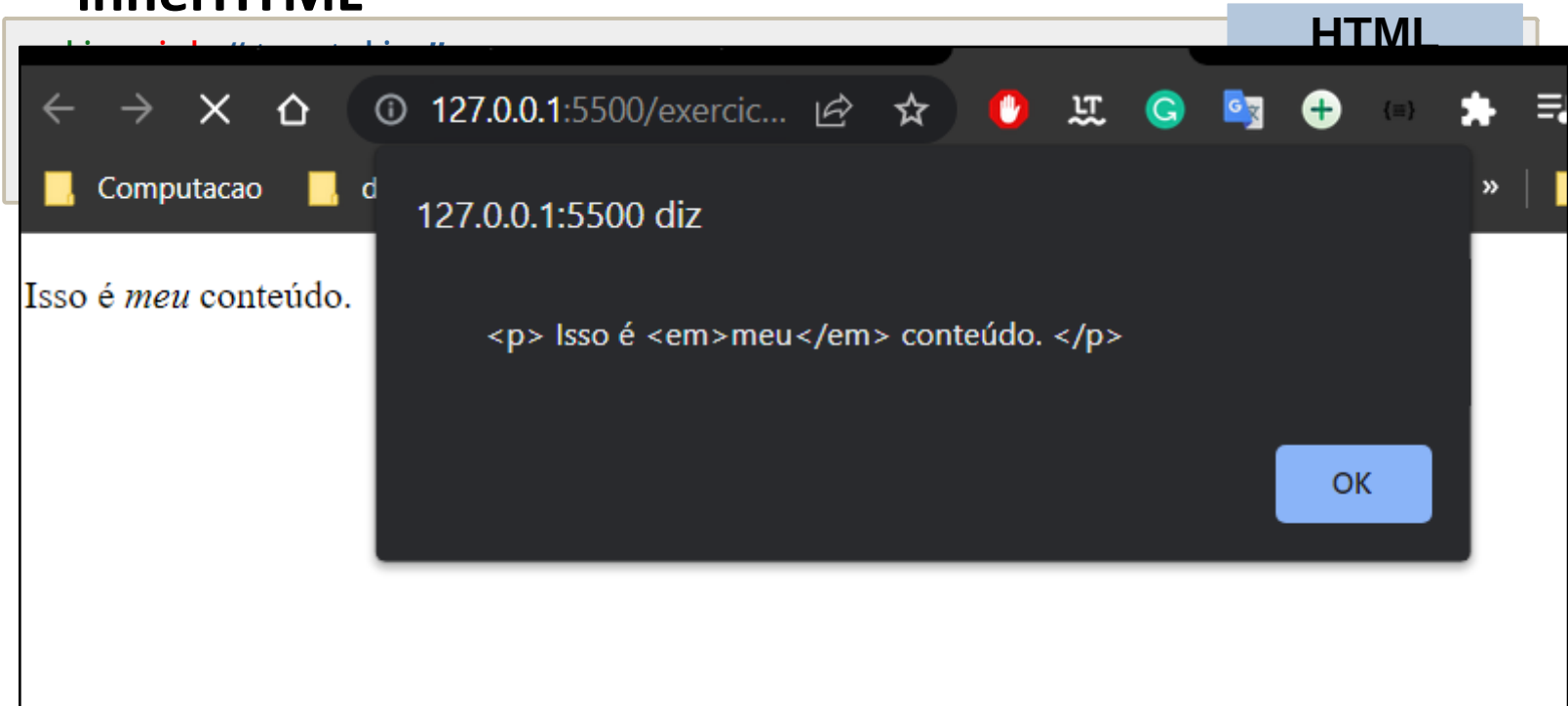
```
window.onload = function(){  
  let testdiv = document.getElementById('testdiv');  
  alert(testdiv.innerHTML);  
}
```

JS



Manipulando árvore DOM

- innerHTML



```
JS
window.onload = function(){
  let testdiv = document.getElementById('testdiv');
  alert(testdiv.innerHTML);
}
```



Manipulando árvore DOM

InnerHTML

- **Prós**
 - ✓ Maneira rápida e fácil de inserir um pedaço de HTML em um documento.
 - ✓ innerHTML se comporta favoravelmente a document.write. Usando innerHTML, você pode manter seu JavaScript separado de sua marcação.
 - ✓ Não há necessidade de inserir tags `<script>` no `<body>` do seu documento.



Manipulando árvore DOM

InnerHTML

- **Contras**

- × É específico do HTML. Você não poderá usá-lo em nenhum outro tipo de documento de marcação.
- × Não retorna nenhuma referência ao conteúdo que você insere. Se você deseja manipular o conteúdo inserido, precisará dos valores oferecidos pelos métodos DOM



Manipulando árvore DOM

- **Abordagem:**
 - São métodos que permitem **criar elementos** e **nó texto** , em seguida, anexá-los ou removê-los na árvore DOM.
- **Adicionar:**
 - Para adicionar conteúdo, você usa um método DOM para criar um novo conteúdo, um nó por vez, e armazená-lo em uma variável. Em seguida, outro método DOM é usado para anexá-lo ao lugar certo na árvore DOM.
- **Remover:**
 - Você pode remover um elemento (junto com qualquer conteúdo e elementos filhos que ele possa conter) da árvore DOM usando um único método



Adicionando um elemento na árvore DOM

- `CreateElement(<tag>)`: você inicia o processo pela criação de um novo elemento. Este elemento é armazenado em uma variável.
- `createTextNode('text')`: **cria** um novo nó texto. Este nó é armazenado em uma variável. Logo após, você pode adicionar no elemento criado, através do método `appendChild()`.
- `AppendChild(HTMLElement)`: agora que você tem seu elemento (opcionalmente com algum conteúdo em um nó de texto), utilize este método para poder adicioná-lo à árvore DOM.

Adicionando um elemento na árvore DOM

HTML

```
<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>
```

JS

```
let text = document.createTextNode('água');
let ele = document.createElement('li');
ele.appendChild(text);
let position = document.getElementsByTagName('ul')[0];
position.appendChild(ele);
```



Elemento na DOM

HTML

Lista de Compras

Comprar Mantimentos

- Picanha fresca
- Castanha
- Mel
- Vinagre

```
let text = document.createTextNode('');
let ele = document.createElement('li');
ele.appendChild(text);
let position = document.getElementsByTagName('li')[0];
position.appendChild(ele);
```

fresca

Lista de Compras

Comprar Mantimentos

- Picanha fresca
- Castanha
- Mel
- Vinagre
- água



Adicionando um elemento na árvore DOM

1: `let text = createTextNode('água')`

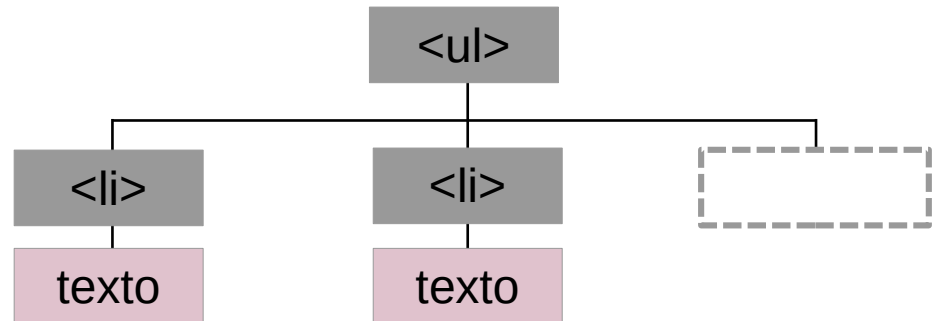
água

2: `let ele = createElement('li')`

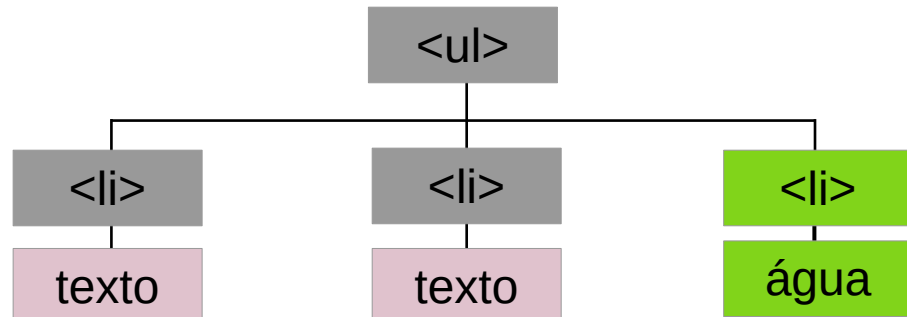
3: `ele.appendChild(text)`

água

4: `let position = getElementByTagName('ul')[0]`

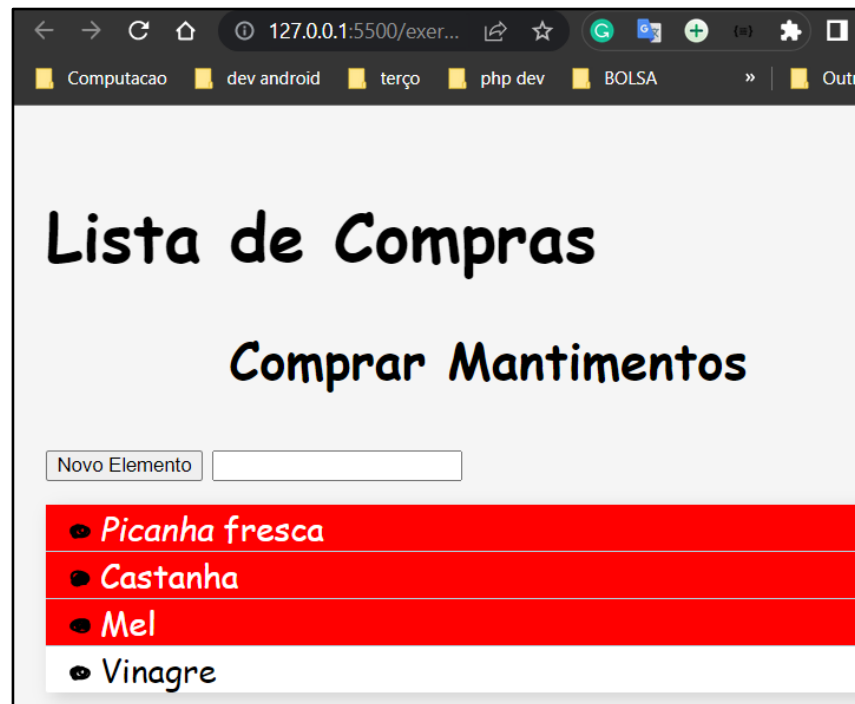


5: `position.appendChild(position)`



Adicionando um elemento na árvore DOM

- **Atividade:** Com base no exemplo anterior, adicione um `<button>` e `<input>` para adicionar um novo elemento na lista de compras de mantimentos. **Conforme resultado abaixo**



Manipulando árvore DOM

- Essas propriedades e métodos estão disponíveis para todos os objetos de elementos
- Adicionalmente, o objeto **document** define dois métodos para criação de novos elementos
 - Esses métodos são essenciais para adicionar conteúdo ao documento HTML

Member	Description	Returns
<code>createElement(<tag>)</code>	Creates a new <code>HTMLElement</code> object with the specific tag type	<code>HTMLElement</code>
<code>createTextNode(<text>)</code>	Creates a new <code>Text</code> object with the specified content	<code>Text</code>



Manipulando árvore DOM

- Todos os elementos da árvore DOM possuem os métodos abaixo, para manipulação da árvore DOM

Member	Description	Returns
<code>appendChild(HTMLElement)</code>	Appends the specified element as a child of the current element	HTMLElement
<code>cloneNode(boolean)</code>	Copies an element	HTMLElement
<code>compareDocumentPosition(HTMLElement)</code>	Determines the relative position of an element	number
<code>innerHTML</code>	Gets or sets the element's contents	string
<code>insertAdjacentHTML(<pos>, <text>)</code>	Inserts HTML relative to the element	void
<code>insertBefore(<newElem>, <childElem>)</code>	Inserts the first element before the second (child) element	HTMLElement



Manipulando árvore DOM

- Todos os elementos da árvore DOM possuem os métodos abaixo, para manipulação da árvore DOM

Member	Description	Returns
<code>isSameNode(HTMLElement)</code>	Determines if the specified element is the same as the current element	boolean
<code>outerHTML</code>	Gets or sets an element's HTML and contents	string
<code>removeChild(HTMLElement)</code>	Removes the specified child of the current element	HTMLElement
<code>replaceChild(HTMLElement, HTMLElement)</code>	Replaces a child of the current element	HTMLElement



Manipulando árvore DOM

- `InsertBefore()`: é um método que permite inserir um nó antes de outro nó a partir de um pai especificado.

HTML

```
<ul id="menu">
  <li>Service</li>
  <li>About</li>
  <li>Contact</li>
</ul>
```

JS

```
//Seleciono o nó elemento PAI
let menu = document.getElementById('#menu');

//criando novo elemento
let li = document.createElement('li');
li.textContent = 'Home';

//Insere novo nó antes do primeiro elemento filho
menu.insertBefore(li, menu.firstChild);
```

Manipulando árvore DOM

- `InsertBefore()`: é um método que permite inserir um nó antes de outro nó a partir de um pai especificado.

HTML

```
<ul id="menu">
  <li>Service</li>
  <li>About</li>
  <li>Contact</li>
</ul>
```

Novo nó (elemento) criado

`HOME`

```
//Seleciono o nó elemento
let menu = document.getElementById('menu');

//criando novo elemento
let li = document.createElement('li');
li.textContent = 'Home';

//Insere novo nó antes do primeiro elemento filho
menu.insertBefore(li, menu.firstChild);
```

Manipulando árvore DOM

- `InsertBefore()`: é um método que permite inserir um nó antes de outro nó a partir de um pai especificado.

HTML

```
<ul id="menu">
  <li>Service</li>
  <li>About</li>
  <li>Contact</li>
```

Se o valor do parâmetro for NULL o elemento será inserido no final da lista

JS

```
document.getElementById('#menu');

//criando novo elemento
let li = document.createElement('li');
li.textContent = 'Home';

//Inserir novo nó antes do primeiro elemento filho
menu.insertBefore(li, menu.firstChild);
```

Manipulando árvore DOM

- **Atividade:** Utilize o método `insertBefore()` para modificar a árvore DOM. Retire ('capture') o primeiro elemento da `List_2` e insira na `List_1`

HTML

```
<ul id="list_1">  
  <li>Café</li>  
  <li>Chá</li>  
</ul>
```

```
<ul id="list_2">  
  <li>Água</li>  
  <li>Leite</li>  
  <li>Suco</li>  
</ul>
```

```
<button onclick="change()">Trocar</button>
```

Manipulando árvore DOM

- `InsertAdjacentHTML()`: é um **método** da **interface Element** para que você possa invocá-lo de qualquer elemento.
- O método analisa um pedaço de texto HTML e insere os nós resultantes na árvore DOM em uma posição especificada
- Parâmetros:
 - Position
 - Text;



Manipulando árvore DOM

- `InsertAdjacentHTML(<position>, <text>)`:
- **POSITION**
 - é uma *string* que representa a posição relativa ao elemento.

Valores	Descrição
beforebegin	Antes (before) do elemento
afterbegin	Antes (before) de seu primeiro filho do elemento
beforeend	após o último filho do elemento
afterend	depois do elemento



Manipulando árvore DOM

- `InsertAdjacentHTML(<position>, <text>)`:
- **POSITION**

```
<h2>Web Technology</h2> ← beforebegin
▼ <ul id="list">
  <li>HTML</li> ← afterbegin
  <li>CSS</li>
  <li>JavaScript</li> ← beforeend
</ul>
<p>For frontend developers</p> ← afterend
```



Manipulando árvore DOM

- `InsertAdjacentHTML(<position>, <text>)`:
- **TEXT**: é uma string que o método `insertAdjacentHTML()` analisa como HTML ou XML. **Não pode ser objetos Node**
- **Consideração de segurança**
 - Como o `innerHTML`, se você usar a entrada do usuário para passar para o método `insertAdjacentHTML()`, você deve sempre evitá-lo para evitar riscos de segurança.



Manipulando árvore DOM

- `InsertAdjacentHTML(<position>, <text>):`

HTML

```
<ul id="list">  
  <li>CSS</li>  
</ul>
```

JS

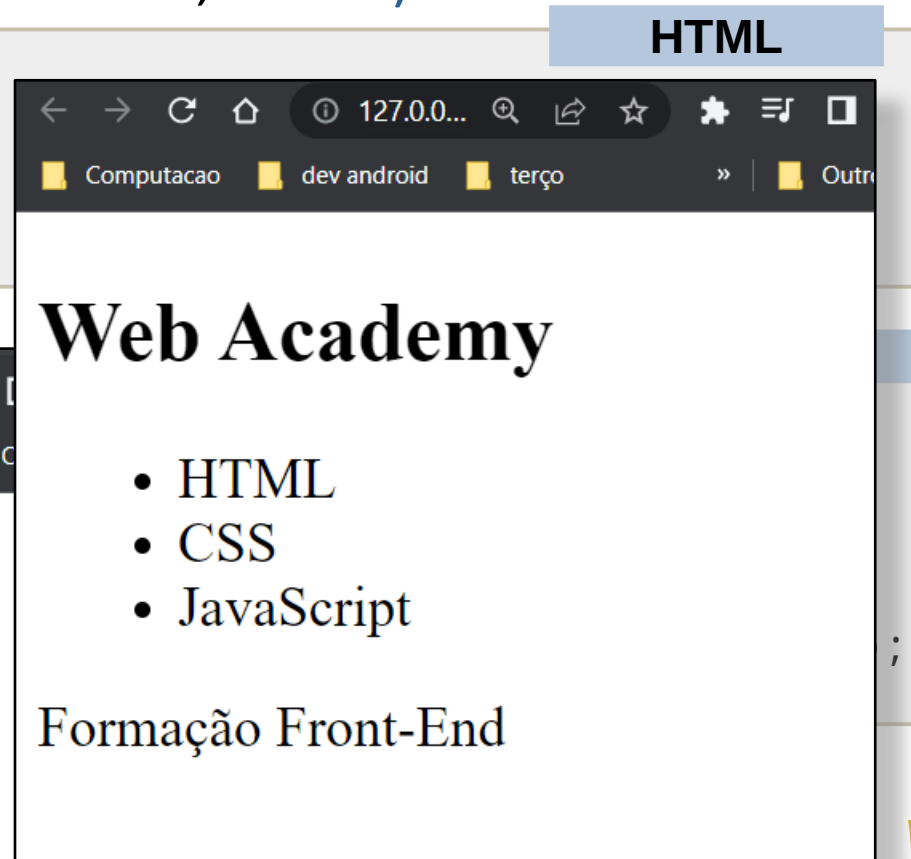
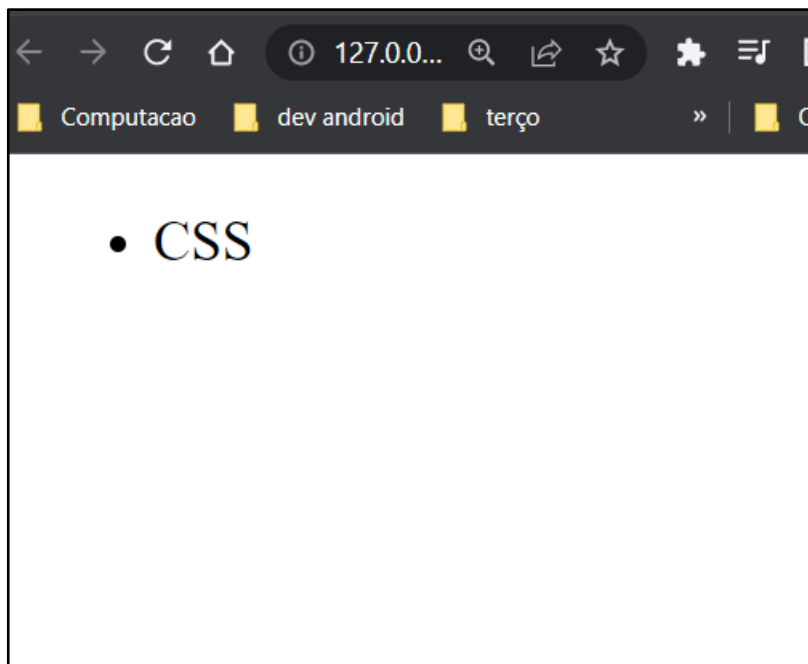
```
let list = document.querySelector('#list');  
list.insertAdjacentHTML('beforebegin', '<h2>Web Academy</h2>');  
list.insertAdjacentHTML('afterbegin', '<li>HTML</li>');  
list.insertAdjacentHTML('beforeend', '<li>JavaScript</li>');  
list.insertAdjacentHTML('afterend', '<p>Formação Front-End</p>');
```



Manipulando árvore DOM

- `InsertAdjacentHTML(<position>, <text>):`

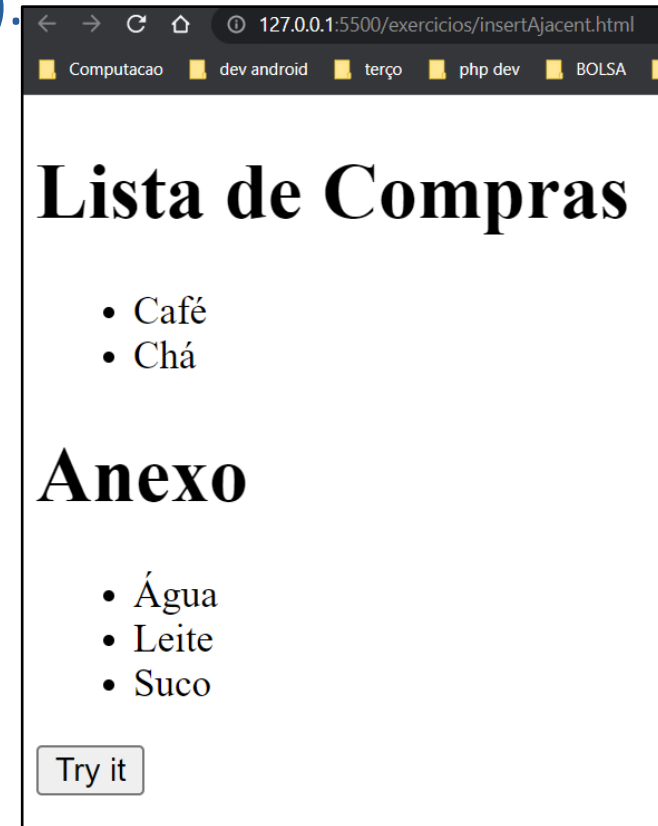
```
<ul id="list">  
  <li>CSS</li>  
</ul>
```



Manipulando árvore DOM

- **Atividade:** Utilizando o HTML SLIDE 20, insira um `<h1> Lista de Compras </h1>` acima do id = list_1 e no final da mesma lista coloque `<h1> Anexo </h1>`. Manipule a DOM, utilizando o método `InsertAdjacentHTML()`.

- Resultado



Manipulando árvore DOM

- `replaceChild(<newElement>, <oldElement>)`
 - Troca um elemento na DOM;
 - Substituir uma tag (Elemento) por outro



Manipulando árvore DOM

- `replaceChild(<newElement>, <oldElement>)`

HTML

```
<ul id="menu">
  <li>HomePage</li>
  <li>Service</li>
  <li>About</li>
  <li>Contact</li>
</ul>
```

Novo nó (elemento) criado

`HOME`

```
//Seleciono o nó elemento PAI
let menu = document.getElementById('#menu');

//criando novo elemento <li></li>
let li = document.createElement('li');
li.textContent = 'Home';

//Troca (replace) usando NOVO nó no lugar do primeiro filho
menu.replaceChild(li, menu.firstChild);
```



Manipulando árvore DOM

- `replaceChild(<newElement>, <oldElement>)`

HTML

```
<ul id="menu">
  <li>HomePage</li>
  <li>Service</li>
  <li>About</li>
  <li>Contact</li>
</ul>
```

O segundo parâmetro corresponde
Qual nó (elemento) irei fazer a troca

JS

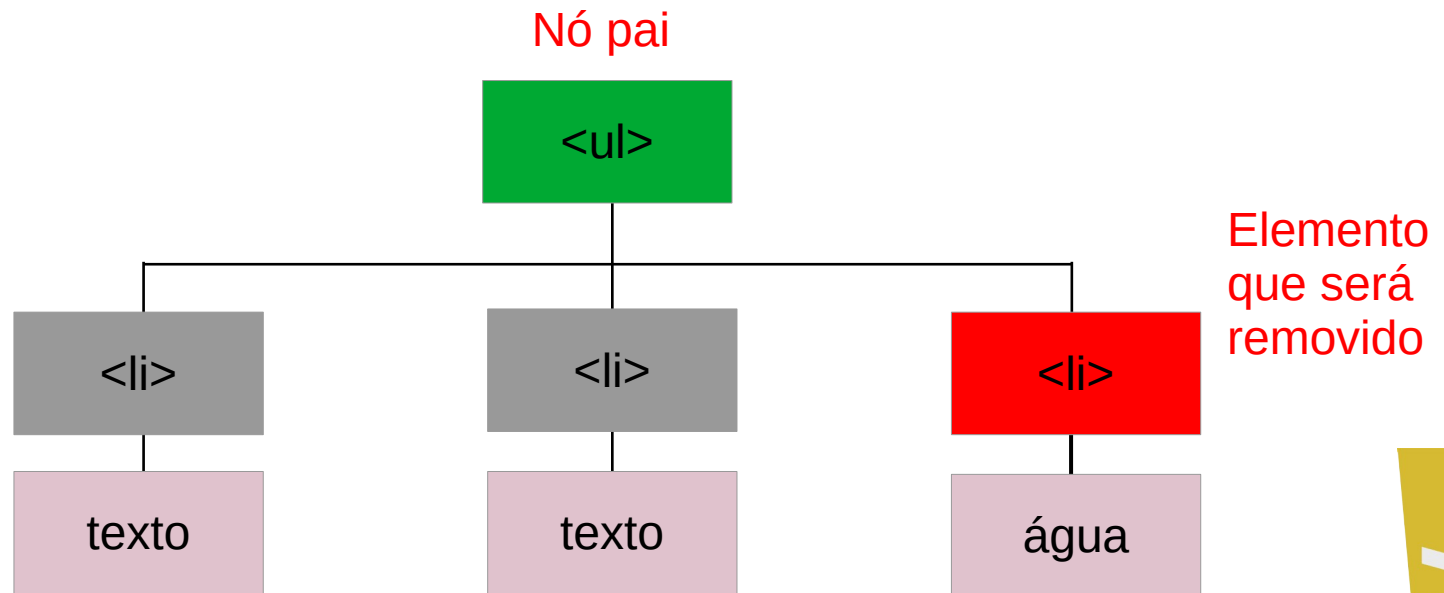
```
//troca o primeiro elemento do menu por um novo elemento
//criando novo elemento
let li = document.createElement('li');
li.textContent = 'Home';

//Troca (replace) usando NOVO nó no lugar do primeiro filho
menu.replaceChild(li, menu.firstChild);
```



Removendo um elemento da árvore DOM

- `removeChild()`: O método remove um nó da árvore DOM. **Tenha cuidado**: este método não é invocado no nó a ser removido, mas no pai desse nó. **Invoque o método no nó pai** e passe o nó filho que deve ser removido como o argumento do método



Removendo um elemento da árvore DOM

HTML

```
<body>
  <div id="page">
    <h1 id="header">Lista de Compras</h1>
    <h2>Comprar Mantimentos</h2>
    <ul id="lista">
      <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>
```

JS

```
let removEle = document.getElementById('lista');

remove.removeChild(removEle.lastChild);
```



Removendo um elemento da árvore DOM

HTML

```
<body>
  <div id="page">
    <h1 id="header">Lista de Compras</h1>
    <h2>Comprar Mantimentos</h2>
    <ul id="lista">
      <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" class="hot"></li>
    </ul>
    <script src="list.js"></script>
  </div>
</body>
```

Nó PAI

Nó FILHO

JS

```
let removEle = document.getElementById('lista');
removEle.removeChild(removEle.lastChild);
```



Removendo um elemento da árvore DOM

HTML

<body>

```
na</em> fresca</li>
/li>
```

Lista de Compras

Comprar Mantimentos

Novo Elemento Remove

- Picanha fresca
- Castanha
- Mel
- Vinagre

```
remove.removeChild(remove.firstChild)
```

127.0.0.1:5500/exer... Computacao dev android terzo php dev BOLSA Out

Lista de Compras

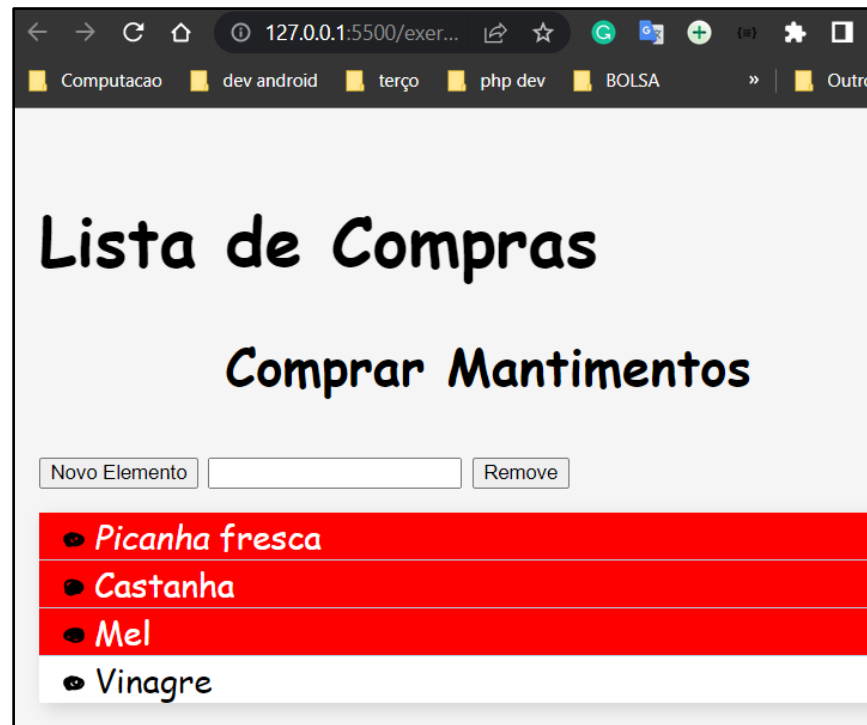
Comprar Mantimentos

Novo Elemento Remove

- Picanha fresca
- Castanha
- Mel

Adicionando um elemento na árvore DOM

- **Atividade:** Com base no exemplo anterior, adicione um `<button>` para **REMOVER** o primeiro elemento na lista de compras de mantimentos. Conforme resultado abaixo



Trabalhando com Eventos

- Os programas JavaScript do lado do cliente usam um modelo de programação orientado a eventos assíncrono
- **Eventos ocorrem** quando os usuários clicam ou tocam em um link, passa ou desliza sobre algum elemento, escreve usando teclado, redimensiona a janela ou quando a página solicitada for carregada
- **Código disparado por Evento:** quando um evento ocorre, dispara alguma função que pode interagir com o usuário por diferentes meios na página.
- **Resposta para o usuário:** os códigos que são disparados podem realizar alterações na DOM. Isso permite dar a sensação de reatividade nas páginas.



Definições Eventos

- **Event type:** é uma string que especifica que tipo de evento ocorreu. Ex: keydown, click, mousemove.
- **Event target:** é o objeto no qual o evento ocorreu ou com o qual o evento é associado. Ex: <button>
- **Event handler** ou **Event Listener:** é uma função que manipula ou responde a um evento. Os aplicativos registram suas funções de manipulador de eventos com o navegador da Web, especificando um tipo de evento (*event type*) e um destino de evento (*event target*)



Definições Eventos

- **Event Object:** é um objeto associado a um evento específico e contém detalhes sobre esse evento. Os *event object* são passados como um argumento para *event handler*. Todos têm uma propriedade de tipo que especifica o tipo de evento e uma propriedade de destino que especifica o *event target*
- **Event Propagation:** é o processo pelo qual o navegador decide quais objetos acionar manipuladores de eventos ativados.



Event Handling

- Quando o usuário interage com o HTML em uma página da Web, há três etapas envolvidas para que ele acione algum código JavaScript. Juntas, essas etapas são conhecidas como **event handling**.

1

Selecionar Elemento

selecione o(s) nó(s) de elemento aos quais deseja que o script responda.

2

Especificar o Evento

indicar qual evento no(s) nó(s) selecionado(s) irá acionar a resposta

3

Call Code

indique o código que deseja executar quando o evento ocorrer



Event Handling

- **event handling** Permite indicar qual evento você está esperando em qualquer elemento específico. Existem três tipos de manipuladores de eventos.

- 1) HTML Event Handlers
- 2) DOM Event Handlers
- 3) DOM Level 2 Event Listeners



HTML Event Handlers

- As primeiras versões do HTML incluíam um conjunto de atributos que podiam responder a eventos no elemento ao qual foram adicionados.
- Seus valores chamavam a função que deveria ser executada quando aquele evento ocorresse.

```
<a onclick =“hide()”> Hide </a>
```

- ✗ Este método de tratamento de eventos não é uma boa prática, porque é melhor separar o JavaScript do HTML.



HTML Event Handlers

- Além da função **addEventListener**, existem outros métodos de se criar manipuladores de eventos
- Por exemplo, podemos usar atributos HTML de acordo com o tipo de evento desejado

HTML

```
<p onmouseover="this.style.background='white';  
this.style.color='black'"  
onmouseout="this.style.removeProperty('color');  
this.style.removeProperty('background')">
```

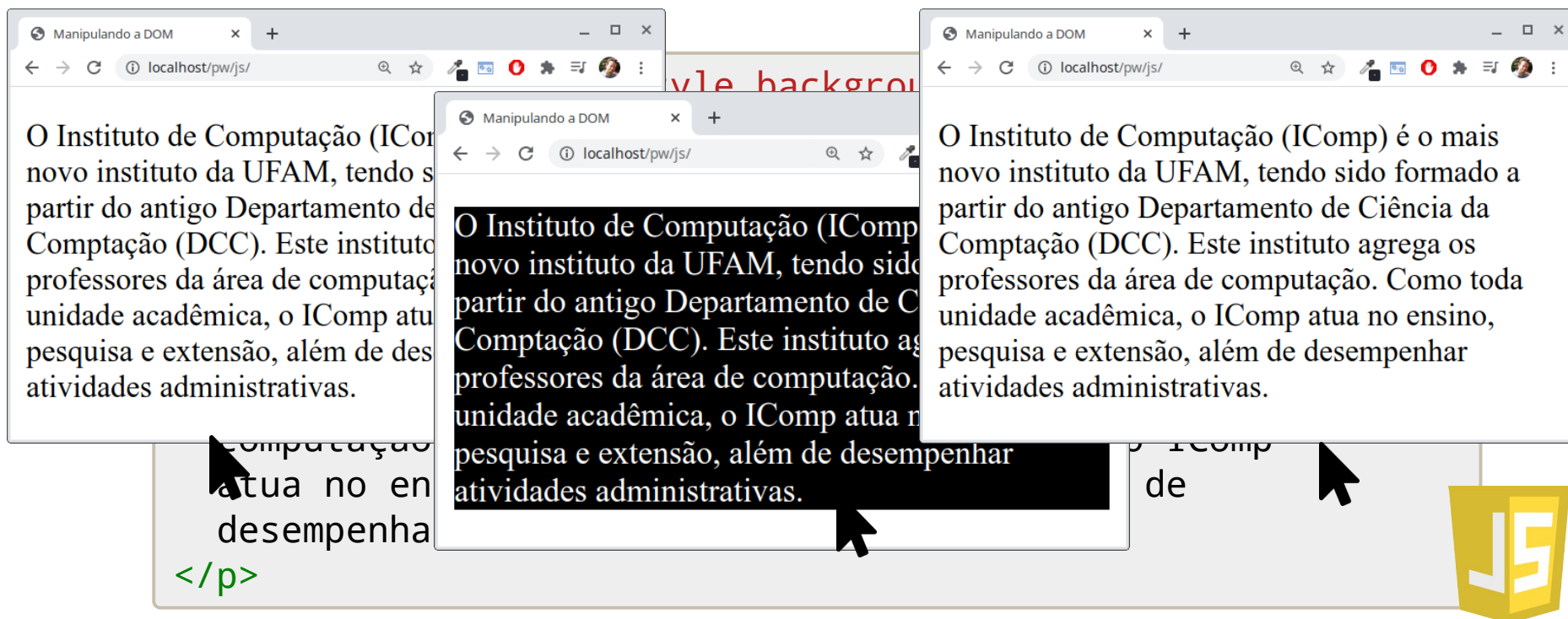
O Instituto de Computação (IComp) é o mais novo instituto da UFAM, tendo sido formado a partir do antigo Departamento de Ciência da Computação (DCC). Este instituto agrega os professores da área de computação. Como toda unidade acadêmica, o IComp atua no ensino, pesquisa e extensão, além de desempenhar atividades administrativas.

```
</p>
```



HTML Event Handlers

- Além da função **addEventListener**, existem outros métodos de se criar manipuladores de eventos
- Por exemplo, podemos usar atributos HTML de acordo com o tipo de evento desejado



HTML Event Handlers

- Também é possível chamar funções em resposta aos eventos

HTML

```
<p onmouseover="handleMouseOver(this)"  
onmouseout="handleMouseOut(this)">
```

O Instituto de Computação (IComp) é o mais novo instituto da UFAM, tendo sido formado a partir do antigo Departamento de Ciência da Computação (DCC). Este instituto agrega os professores da área de computação.

```
</p>
```

JS

```
function handleMouseOver(elem) {  
    elem.style.background='black';  
    elem.style.color='white';  
}  
function handleMouseOut(elem) {  
    elem.style.removeProperty('color');  
    elem.style.removeProperty('background');  
}
```



DOM Event Handling

- Foram introduzidos na especificação original para o DOM. Eles são considerados melhores do que os manipuladores de eventos HTML porque permitem separar o JavaScript do HTML
- ✗ A principal desvantagem é que você só pode anexar uma única função a qualquer evento.
- ✗ Se mais de um script for usado na mesma página e ambos os scripts responderem ao mesmo evento, um ou ambos os scripts podem não funcionar conforme o esperado.



DOM 2 Event Handling

`element.onevent = functionName;`

Elemento nó

Event

Code

JS

```
function checkUserName(){
    //code to check length of username
}

//elemento de algum campo do formulário
var el = document.getElementById('username');

//adiciona o evento no elemento
el.onblur = checkUserName; //functionName
```



Tipos de Eventos

- ***User interface events***: são eventos de alto nível, geralmente são aplicados em elementos de formulário HTML.

Eventos de formulários

Evento	Descrição
onblur	Evento disparado quando o elemento perde o foco, perde a seleção.
onchange	Evento disparado quando o conteúdo do elemento é alterado, input, keygen, select, textarea.
onfocus	Evento disparado quando o elemento recebe o foco, é selecionado.
onfocusin	Evento disparado quando o elemento está prestes a receber o foco.
onfocusout	Evento disparado quando o elemento está prestes a perder o foco.
oninput	Evento disparado quando o usuário clica para entrar ou selecionar o elemento.
oninvalid	Evento disparado quando um elemento é inválido.
onreset	Evento disparado quando o botão reset é pressionado.
onsearch	Evento disparado quando é escrito algo em um campo <input type="search">
onselect	Evento disparado logo após a seleção de um texto <text> e <textarea>
onsubmit	Evento disparado quando o botão submit é clicado para enviar os dados do formulário.



Tipos de Eventos

- ***Dependent input events***: são eventos diretamente vinculados a um dispositivo de entrada específico, como o mouse ou o teclado

Eventos de mouse

Evento	Descrição
onclick	Evento disparado quando se clica em um elemento.
oncontextmenu	Evento disparado quando se clica com o botão direito do mouse sobre um elemento.
ondblclick	Evento disparado quando é aplicado um clique duplo sobre um elemento.
onmousedown	Evento disparado quando o botão do mouse é pressionado sobre um elemento.
onmouseenter	Evento disparado quando o ponteiro do mouse se move para cima de um elemento.
onmouseleave	Evento disparado quando o ponteiro do mouse se move para fora de um elemento.
onmousemove	Evento disparado quando o ponteiro do mouse se move sobre um elemento.
onmouseover	Evento disparado quando o ponteiro do mouse é movido para dentro de um elemento ou um de seus filhos.
onmouseout	Evento disparado quando o ponteiro do mouse é movido para fora de um elemento ou um de seus filhos.
onmouseup	Evento disparado quando o botão do mouse é liberado, desprecando, sobre um elemento.



Tipos de Eventos

- ***Dependent input events***: são eventos diretamente vinculados a um dispositivo de entrada específico, como o mouse ou o teclado

Eventos de teclado

Evento	Descrição
onkeydown	Evento disparado quando o usuário pressiona uma tecla.
onkeypress	Evento disparado quando o usuário mantém uma tecla pressionada.
onkeyup	Evento disparado quando o usuário libera uma tecla pressionada.



Tipos de Eventos

- ***Independent input events***: são eventos que não estão diretamente vinculados a um dispositivo de entrada específico.
- O evento click, por exemplo, indica que um link ou botão (ou outro elemento do documento) foi ativado. Isso geralmente é feito por meio de um clique do mouse, mas também pode ser feito pelo teclado ou (em dispositivos sensíveis ao toque) por gestos.



Tipos de Eventos

- ***Stage-change events***: Alguns eventos não são acionados diretamente pela atividade do usuário, mas pela atividade da rede ou do navegador, e indicam algum tipo de ciclo de vida ou alteração relacionada ao estado

Evento	Descrição
onmessage	Evento disparado quando uma mensagem é recebida através ou a partir de um objeto.
ononline	Evento disparado quando o browser inicia o trabalho online.
onoffline	Evento disparado quando o browser inicia o trabalho offline.
onpopstate	Evento disparado quando a janela de histórico muda.
onshow	Evento disparado quando um elemento de <menu> é mostrado.
onstorage	Evento disparado quando a área de armazenamento web é atualizada.
ontoggle	Evento disparado quando o usuário abre ou fecha um elemento de <details>
onwheel	Evento disparado quando a roda do mouse é usada.



Tipos de Eventos

- **API-specific events:** Várias APIs da web definidas pelo HTML5 e especificações relacionadas incluem seus próprios tipos de eventos.

Eventos de Drag / Arrastar

Evento	Descrição
ondrag	Evento disparado quando um elemento é arrastado.
ondragend	Evento disparado quando um elemento deixa de ser arrastado.
ondragenter	Evento disparado quando o elemento arrastado entra no elemento alvo.
ondragleave	Evento disparado quando o elemento arrastado deixa o elemento alvo.
ondragover	Evento disparado quando o elemento arrastado está sobre o elemento alvo.
ondragstart	Evento disparado quando o elemento começa a ser arrastado.
ondrop	Evento disparado quando o elemento arrastado é solto.



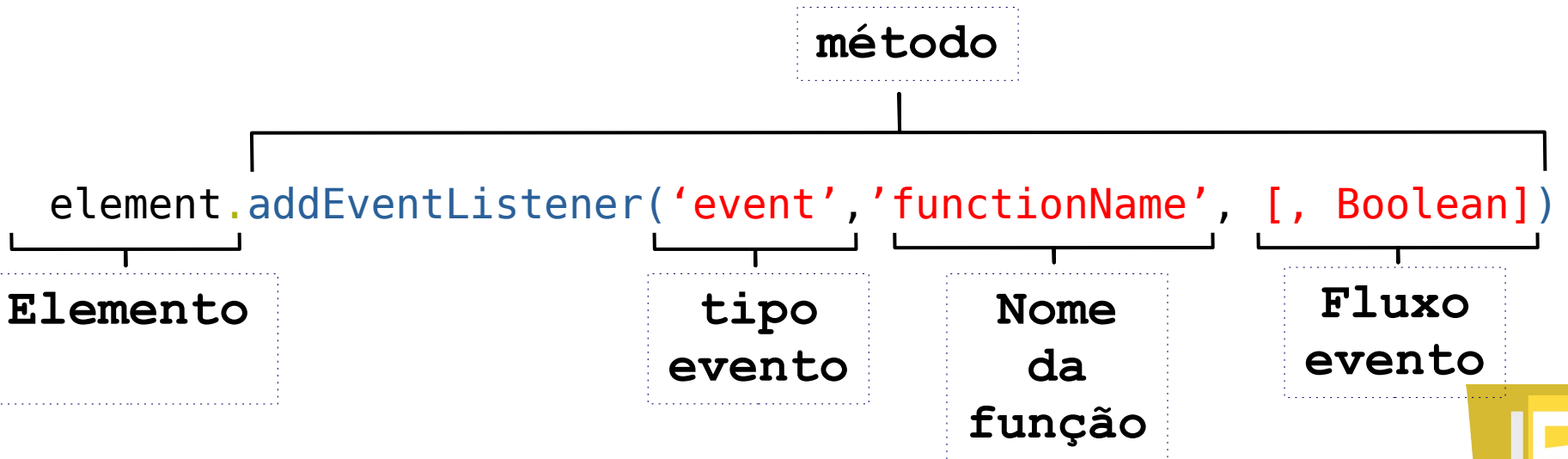
DOM Level 2 Event Handlers

- Permitem que um evento acione várias funções. Como resultado, é menos provável que haja conflitos entre diferentes scripts executados na mesma página.
- **DOM Level 3:** Esse novo padrão DOM também simplifica os eventos `keydown`, `keyup` e `keypress` adicionando novas propriedades de chave e caractere ao objeto de evento. Ambas as propriedades são strings. Para eventos de chave que geram caracteres imprimíveis, `key` e `char` serão iguais ao texto gerado.



Event Listeners

- É uma abordagem mais recente para lidar com eventos.
- Podem lidar com mais de uma função ao mesmo tempo.
- Define um método chamado `addEventListener()`



Event Listeners

HTML

```
<form id="myform">
  <label for="username">Create a username: </label>
  <input type="text" id="username" />
  <div id="feedback"></div>
  <label for="password">Create a password: </label>
  <input type="password" id="password" />
  <input type="submit" value="Sign up!" />
</form>
```

JS

```
let userName = document.getElementById('username');
userName.addEventListener('blur', checkUserName);

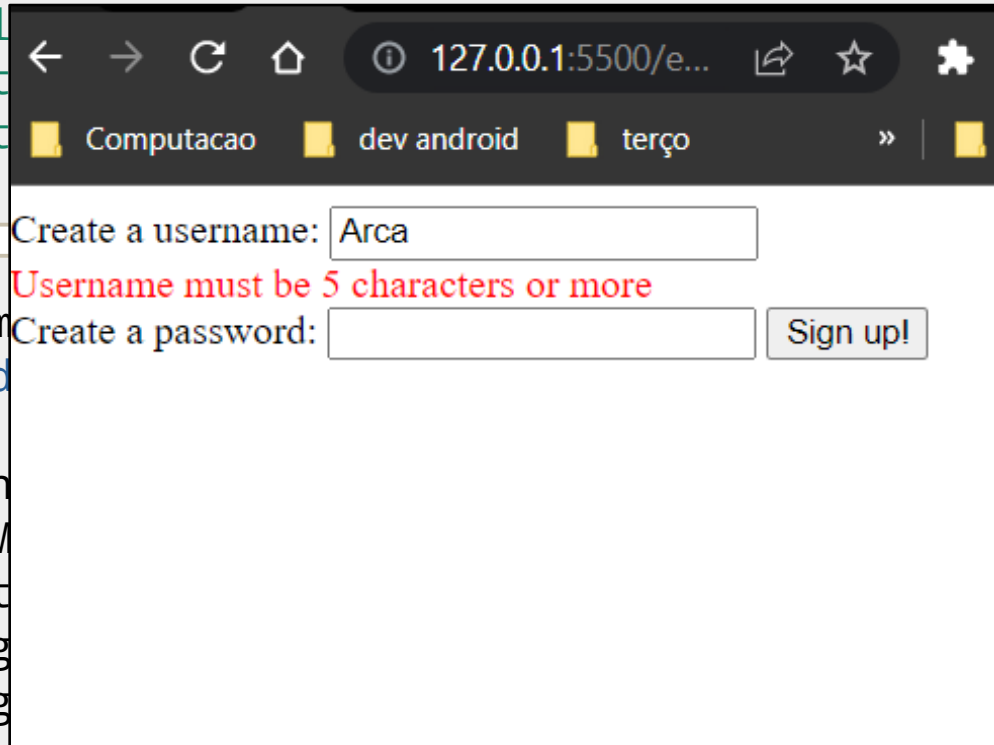
function checkUserName(){
  let elMsg = document.getElementById('feedback');
  if(this.value.length < 5){
    elMsg.textContent = 'Username must be 5 characters or more';
    elMsg.style.cssText = "color:red";
  }else{ elMsg.textContent = ''; }
}
```



Event Listeners

HTML

```
<form id="myform">
  <label for="username">Create a username: </label>
  <input type="text" id="username" />
  <div id="feedback"></div>
  <label
  <input
  <input
</form>
```



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:5500/e...'. The browser has tabs for 'Computacao', 'dev android', and 'terço'. The form in the browser has a 'Create a username:' label followed by a text input field containing 'Arca'. Below this is a red error message: 'Username must be 5 characters or more'. Then there is a 'Create a password:' label followed by a text input field. To the right of the password field is a 'Sign up!' button.

```
let userName
userName.ad

function ch
  let elm
  if(el.t
    elMsg
    elMsg
  }else{
    elMsg.textContent = ;
  }
```

JS

acters or more';

Tipos de Eventos

- Carregamento de página (UI Event)
 - **load**: é acionado quando a página da Web termina de carregar. Ele também pode disparar em nós de elementos como imagens, scripts ou objetos.
 - **unload**: é acionado antes que os usuários saiam da página.
 - **error**: é acionado quando o navegador encontra um erro de JavaScript ou um ativo que não existe.
 - **resize**: Ele é acionado quando redimensionamos a janela do navegador. Mas os navegadores acionam repetidamente esse evento
 - **scroll**: é disparado quando o usuário rola para cima/para baixo na janela do navegador.



Tipos de Eventos

- Elementos em foco
 - **focus:** é acionado, para um nó DOM específico, quando um elemento ganha foco.
 - **blur:** é acionado, para um nó DOM específico, quando um elemento perde o foco



Tipos de Eventos

- Eventos do mouse (*Mouse event*)
 - **click**: é disparado quando o usuário clica no botão principal do mouse. também é acionado se o usuário pressionar a tecla Enter no teclado quando um elemento estiver em foco.
 - **dblclick**: é acionado quando o usuário clica no botão principal do mouse, em rápida sucessão, duas vezes.
 - **mousedown**: é acionado quando o usuário clica em qualquer botão do mouse.
 - **mouseup**: é acionado quando o usuário libera um botão do mouse.



Eventos de mouse

- Cada manipulador de evento é registrado para um dado elemento HTML, tal como **window** no exemplo anterior
- Cada elemento da árvore DOM possui seu próprio método **addEventListener()**
 - Desta forma, todos os elementos podem ser “ouvidos”

```
<button>Me clique!</button>
<script>
  let button = document.querySelector("button");
  button.addEventListener("click", function() {
    console.log("Botão clicado!");
  });
</script>
```



Eventos de mouse

- Por exemplo, para saber qual botão foi clicado em um evento **mousedown**, podemos recorrer a esse objeto

```
<button>Me clique!</button>

<script>
  let button = document.querySelector("button");
  button.addEventListener("mousedown", function(event) {
    if (event.which == 1)
      console.log("Botão esquerdo");
    else if (event.which == 3)
      console.log("Botão direito");
  });
</script>
```



Eventos de mouse

- Movimentos do mouse
 - **mouseover**: dispara quando passamos o cursor sobre o elemento.
 - **mouseout**: dispara quando o cursor sai do elemento.

Os eventos **mouseover** e **mouseout** geralmente alteram a aparência dos gráficos em nossa página da web. Uma alternativa preferida para isso é usar o **CSS: hover** pseudo-classe.

- **mousemove**: acionado quando o usuário move o cursor ao redor do elemento. (CUIDADO!) **Este evento é frequentemente acionado.**



Eventos de Mouse

- **mousedown** e **mouseup** são similares aos eventos **keydown** e **keyup**, e são disparados no clique do mouse

HTML

```
body {  
    height: 200px; background: beige;  
}  
.dot {  
    height: 8px; width: 8px; border-radius: 4px;  
    background: blue; position: absolute;  
}
```

JS

```
document.addEventListener("mousedown", function(event) {  
    let dot = document.createElement("div");  
    dot.className = "dot";  
    dot.style.left = (event.pageX - 4) + "px";  
    dot.style.top = (event.pageY - 4) + "px";  
    document.body.appendChild(dot);  
});
```

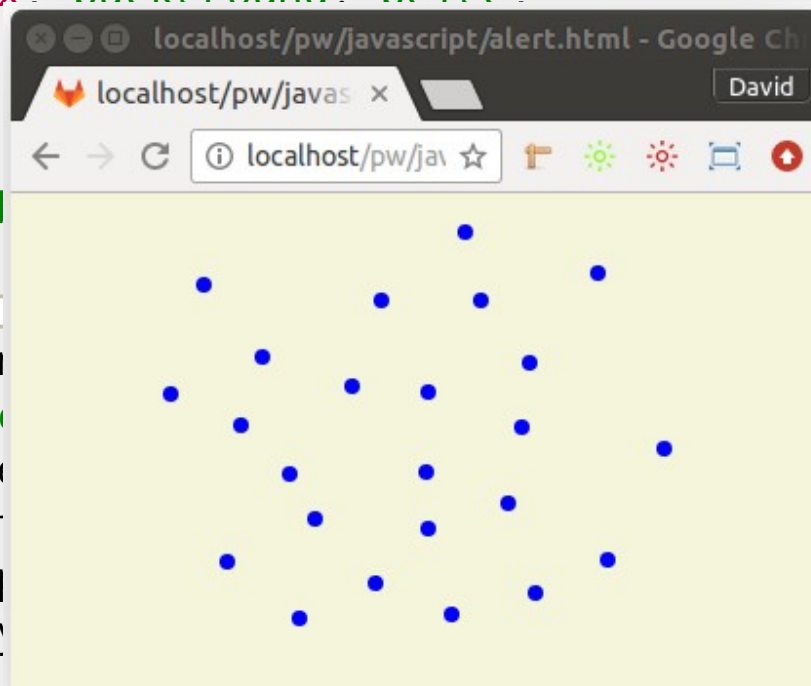


Eventos de Mouse

- **mousedown** e **mouseup** são similares aos eventos **keydown** e **keyup**, e são disparados no clique do mouse

```
body {  
    height: 200px; background: height:  
}  
.dot {  
    height: 8px;  
    background: l  
}
```

```
document.addEventListener(  
    let dot = doc  
    dot.className  
    dot.style.le  
    dot.style.top  
    document.body  
));  
event) {
```



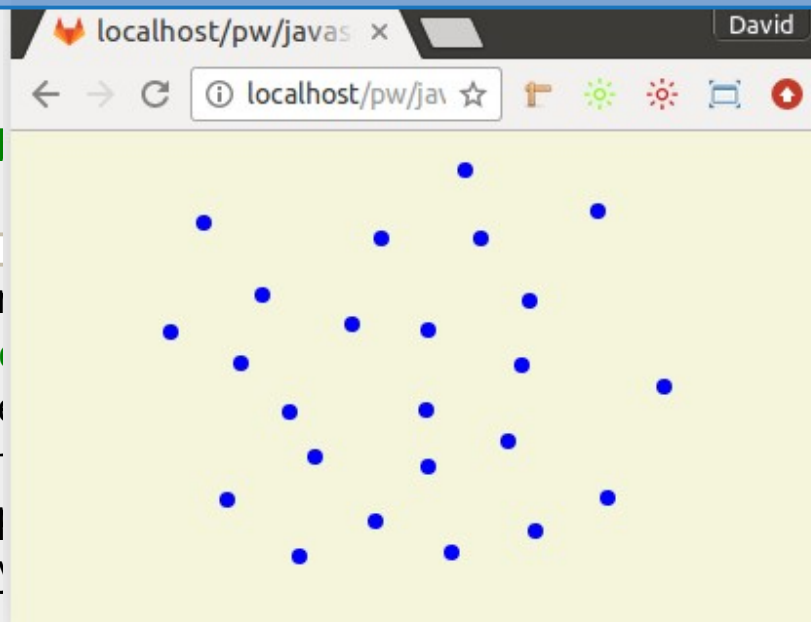
Eventos de Mouse

- **mousedown** e **mouseup** são similares aos eventos **keydown** e **keyup**, e são disparados no clique do mouse

Além de **mousedown** e **mouseup**, também existem os eventos **click** e **dblclick**

```
.dot {  
  height: 8px;  
  background: #000;  
}
```

```
document.addEventListener(  
  'click', (event) => {  
    let dot = document.createElement('div');  
    dot.className = 'dot';  
    dot.style.left = event.clientX + 'px';  
    dot.style.top = event.clientY + 'px';  
    document.body.appendChild(dot);  
  });
```



```
event) {
```



Outros Tipos de Eventos

- Eventos do teclado (*Keyboard Event*)
 - **input**: Este evento é acionado quando o valor de um `<input>` ou `<textarea>` muda
 - **keydown**: dispara quando o usuário pressiona qualquer tecla no teclado.
 - **keypress**: dispara quando o usuário pressiona uma tecla que resulta na impressão de um caractere na tela. Não será acionado para as teclas enter, tab ou seta; use **keydown**.
 - **keyup**: é acionado quando o usuário libera uma tecla no teclado.

Para saber a tecla pressionada quando você usa os eventos `keydown` e `keypress`, possui um **objeto de evento** **keyCode**



Eventos de Teclado

- O browser dispara um evento **keydown** quando uma tecla é clicada, e **keyup** quando a tecla é solta

```
<p>Esta página fica violeta quando V é pressionado</p>
```

```
addEventListener("keydown", function(event) {  
    if (event.keyCode == 86)  
        document.body.style.background = "violet";  
});  
  
addEventListener("keyup", function(event) {  
    if (event.keyCode == 86)  
        document.body.style.background = "";  
});
```



Eventos de Teclado

- Cliques em teclas tais como **Shift**, **Ctrl**, **Alt**, e **Meta** (Command do Mac) também geram eventos
- As propriedades **shiftKey**, **ctrlKey**, **altKey**, e **metaKey** são usadas para identificar tais cliques

```
<p>Pressione Ctrl-Space para continuar.</p>
```

```
addEventListener("keydown", function(event) {  
    if (event.keyCode == 32 && event.ctrlKey)  
        console.log("Continuando!");  
});
```



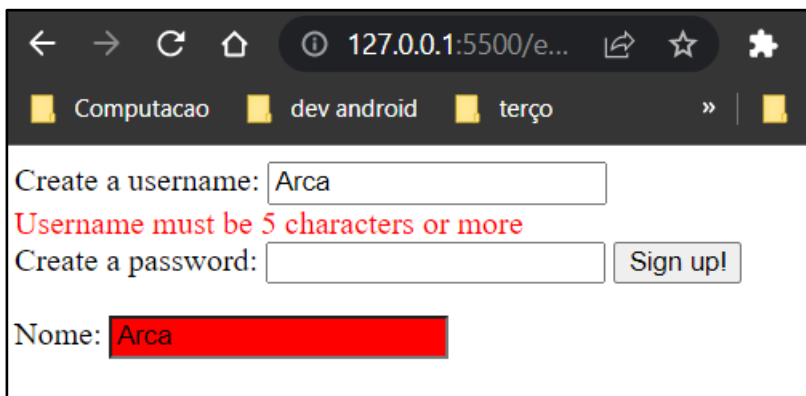
Outros Tipos de Eventos

- Formulários (Form events)
 - **submit:** é acionado no nó que representa o elemento `<form>` quando um usuário envia um formulário.
 - **change:** é acionado quando o status de vários elementos de formulário é alterado. Essa é uma opção melhor do que usar o evento click porque clicar não é a única maneira de os usuários interagirem com o formulário.
 - **input:** é muito comum com os elementos `<input>` e `<textarea>`. Frequentemente usamos os eventos de foco e desfoque com formulários, mas eles também estão disponíveis em conjunto com outros elementos, como links.



Eventos de formulário

- **Atividade:** utilizando o código do slide 38, crie um `<input>` com `id="result"` no formulário e um evento que coloque o mesmo texto do `<input>` username, no input criado e mude a cor para "red" quando for menor que 5 caracteres, e "green" caso contrário. E deixe "white" quando não houver nenhum caractere.



A screenshot of a web browser window showing a form. The address bar displays "127.0.0.1:5500/e...". The browser tabs include "Computacao", "dev android", and "terço". The form contains the following elements:

- A label "Create a username:" followed by an input field containing the text "Arca".
- A red error message below the input: "Username must be 5 characters or more".
- A label "Create a password:" followed by an empty input field.
- A "Sign up!" button.
- A label "Nome:" followed by a red input field containing the text "Arca".



A screenshot of a web browser window showing the same form as the previous image, but with the username "Arcanjo" entered. The error message is gone, and the "Nome:" input field is now green.

- A label "Create a username:" followed by an input field containing the text "Arcanjo".
- A label "Create a password:" followed by an empty input field.
- A "Sign up!" button.
- A label "Nome:" followed by a green input field containing the text "Arcanjo".

O objeto de evento

- **Quando ocorre um evento**, o objeto de evento fornece informações sobre o evento e o elemento no qual ele ocorreu.
- **Fornece dados** úteis sobre o evento, como:
 - Em qual elemento o evento aconteceu
 - Qual tecla foi pressionada
 - Em que parte da viewport o usuário clicou



O objeto de evento

- Propriedades:
 - **target**: Retorna o elemento que acionou o evento
 - **type**: Retorna o nome do evento
 - **cancelable**: Retorna se um evento pode ou não ter sua ação impedida.
- Métodos
 - **preventDefault()**: Cancela o evento se for **cancelable == 'true'**, significando que a ação padrão que pertence ao evento não ocorrerá
 - **stopPropagation()**: Impede a propagação adicional de um evento durante o fluxo do evento



O objeto de evento

HTML

```
<form id="myform">
  <label for="username">Create a username: </label>
  <input type="text" id="username" />
  <div id="feedback"></div>
  <label for="password">Create a password: </label>
  <input type="password" id="password" />
  <input type="submit" value="Sign up!" />
</form>
```

JS

```
let userName = document.getElementById('username');
userName.addEventListener('blur', checkUserName);

function checkUserName(event){
  let userTarget = event.target; //<input type="text" id="username">
  let eventType = event.type; // blur
  let eventCancel = event.cancelable; //false
  let getValueEl = event.target.value // retorna valor do campo
}
```



O objeto de evento

HTML

```
<form id="myform">
  <label for="username">Create a username: </label>
  <input type="text" id="username" />
  <div id="feedback"></div>
  <label for="password">Create a password: </label>
  <input type="password" id="password" />
  <input type="submit" value="Sign up" />
</form>
```

Passando o evento

JS

```
let userName = document.getElementById('username');
userName.addEventListener('blur', checkUserName);

function checkUserName(event){
  let userTarget = event.target; //<input type="text" id="username">
  let eventType = event.type; // blur
  let eventCancel = event.cancelable; //false
  let getValueEl = event.target.value // retorna valor do campo
}
```



Mudando Comportamento Padrão

- `preventDefault()` é um método que garante que a ação padrão não seja executada

```
<a href="http://icomp.ufam.edu.br/" target="_blank">ICOMP</a>
```

```
let link = document.querySelector("a");
link.addEventListener("click", function(event) {
    if(event.cancelable == true){ //event.preventDefault()
        console.log("Usuário não será encaminhado");
        event.preventDefault();
    }
});
```

- Isto pode ser usado, por exemplo, para criar seus próprios atalhos de teclados ou menus de contexto ou desativar botões em caso de alguns erros.



Mudando Comportamento Padrão

- **Atividade:** A partir do seguinte formulário:

```
<form id="myform">
  <label for="username">Create a username: </label>
  <input type="text" id="username" />
  <div id="feedback"></div>
  <label for="password">Create a password: </label>
  <input type="password" id="password" />
  <input type="submit" value="Sign up!" />
</form>
```

- Faça o bloqueio do <button> 'submit' caso o <input> 'username' contenha letras minúsculas e depois mande um `window.alert()` para usuário informando o problema.



Mudando Comportamento Padrão

- `stopPropagation()` método que você pode invocar para impedir a propagação contínua do evento.
- O método pode ser chamado a qualquer momento durante a propagação do evento.
- Funciona durante a fase de captura, no próprio alvo do evento e durante a fase de borbulhamento.



Fluxo do Evento

- Os elementos HTML se aninham dentro de outro elemento. Se você passar o mouse ou clicar em um link, também passará o mouse ou clicará nos elementos pai.
- **Imagine** uma lista de itens que contém um link `<a>`. Quando passar o mouse ou clicar sobre ele, o JavaScript dispara um evento no elemento `<a>` e em quaisquer elemento que `<a>` seja filho.

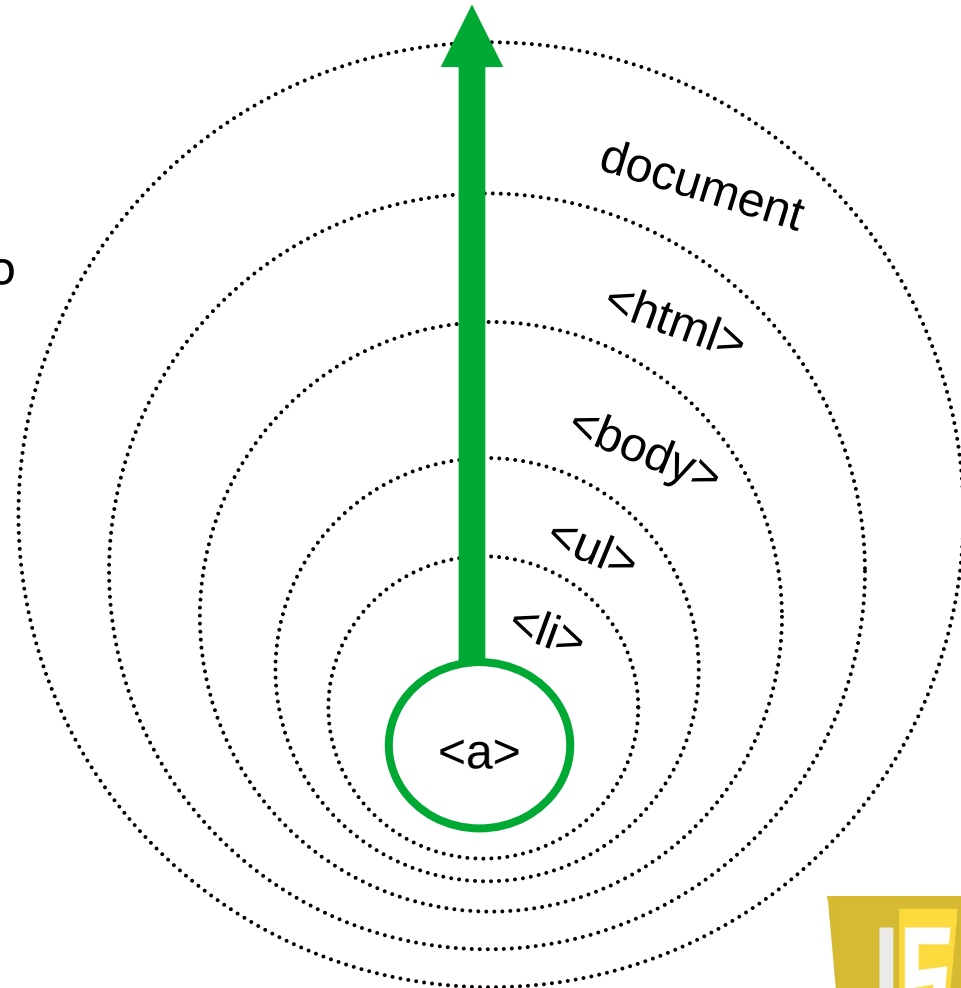


Fluxo do Evento

- **Evento Borbulhamento**

Este evento inicia do nó mais específico e flui 'para fora' para o nó menos específico

No momento, todos os navegadores modernos têm o borbulhamento de eventos como a forma padrão de fluxo de eventos.



Fluxo do Evento

- **Evento Borbulhamento (Exemplo)**

```
<div id="pai">
  <button>
    <h2>Parent</h2>
  </button>
  <button id="filho">
    <p>Child</p>
  </button>
</div><br>
```

```
document.getElementById("filho")
  .addEventListener("click", function () {
    alert("You clicked the Child element!");
  }, false);
document.getElementById("pai")
  .addEventListener("click", function () {
    alert("You clicked the parent element!");
  }, false);
```



Fluxo do Evento

- **Evento Borbulhamento**

- Vimos que o efeito bubbling percorre elementos aninhados no DOM de baixo para cima (do elemento filho para seus ancestrais).
- É possível interromper o efeito bubbling antes que ele percorra todos os elementos aninhados.
- O código para interromper o efeito em navegadores em conformidade com o W3C é mostrado a seguir:

```
element.addEventListener('click', function(event) {  
  if (event.stopPropagation) { //se verdade  
    // opção para padrões W3C  
    event.stopPropagation()  
  }  
});
```



Fluxo do Evento

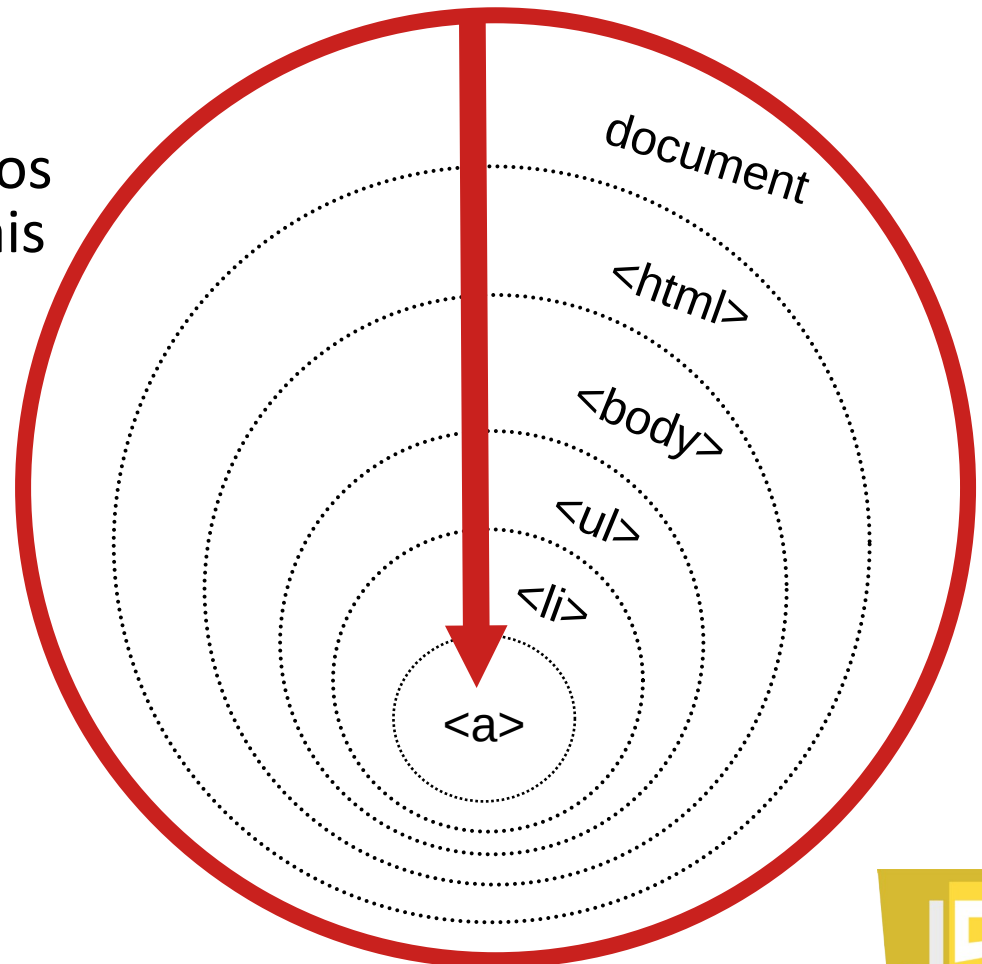
- **Evento Borbulhamento**
 - Como funciona na prática?
 - Se a um elemento for atrelado vários handlers disparados pelo mesmo evento, os handlers serão independentes.
 - Por exemplo: se em um link existirem dois handlers disparados por click, interromper o efeito bubbling em um dos handlers não interrompe no outro. O navegador não tem qualquer compromisso com a ordem de disparo dos handlers.



Fluxo do Evento

- **Captura de evento**

O evento começa no nó menos específico e flui para o nó mais específico.



Fluxo do Evento

- **Captura de evento**
 - Todos os métodos de manipulação de eventos simplesmente ignoram a fase capturing. Para que o evento ocorra na fase capturing declaramos o último argumento método `addEventListener()` como sendo **true**.
 - Na prática a fase capturing raramente é usada, mas existem eventos que não “borbulham”, mas honram o efeito capturing. Por exemplo: `onfocus` e `onblur`.



Fluxo do Evento

- **Atividade:** A partir do código abaixo faça:

HTML

```
<div id="the_div">
  <ul id="the_list">
    <li id="the_item">Click!</li>
  </ul>
</div>

<p id="result"></p>
```

JS

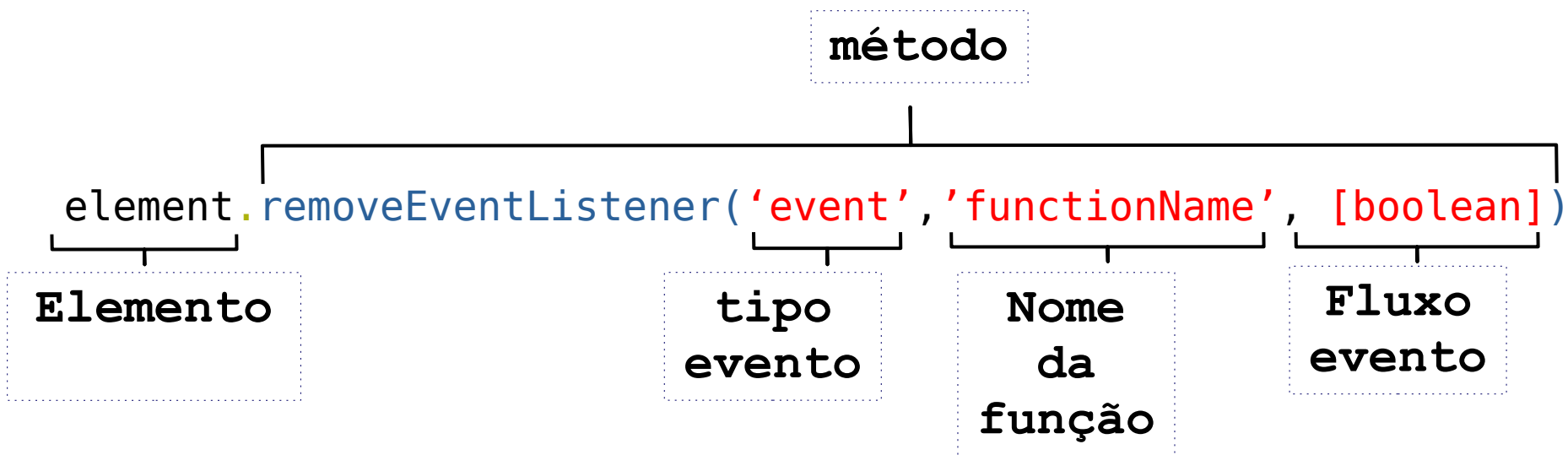
```
function result(strig){
  document.getElementById('result')
    .innerHTML += string + '<br/>';
}
```

- 1) Crie chamadas `addEventListener` para que os eventos ocorram na seguinte ordem: `the_div` → `the_list` → `the_item`
- 2) Altere as chamadas `addEventListener` para que os eventos ocorram na seguinte ordem: `the_item`!



Remover Evento

- `removeEventListener()`: é uma função embutida que é usada para remover um manipulador de eventos que **foi adicionado anteriormente** usando a função `addEventListener()` do elemento.



Remover Evento

- `removeEventListener()`

HTML

```
<button id="save">Salvar</li>
```

JS

```
let btn = document.getElementById('save');

function fnClick(){
  alert('Botao salvar foi clicado');
  button.disable = true; //desabilita botão

  //remove o evento de click do botao save
  button.removeEventListener('click', fnClick);
}

button.addEventListener('click', fnClick);
```



Remover Evento

- `removeEventListener()`

HTML

```
<button id="save">Salvar</li>
```

Propriedade que
'desativa' o botão, mas não
seu evento de click

```
let btn = document.getElementById('save');

function fnClick(){
    alert('Botao salvar foi clicado');
    button.disable = true; //desabilita botão

    //remove o evento de click do botao save
    button.removeEventListener('click', fnClick);
}

button.addEventListener('click', fnClick);
```



RESUMO

- Os eventos são a maneira do navegador indicar quando algo aconteceu (como quando uma página terminou de carregar ou um botão foi clicado).
- ***Binding*** é o processo de declarar qual evento você está esperando para acontecer e qual elemento você está esperando para que esse evento aconteça.
- Quando um evento ocorre em um elemento, ele pode acionar uma função JavaScript. Quando essa função altera a página da Web de alguma forma, ela parece interativa porque respondeu ao usuário.
- Você pode usar a Fluxo de eventos para monitorar os eventos que ocorrem em todos os filhos de um elemento.

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.