Javascript

Introdução ao DOM

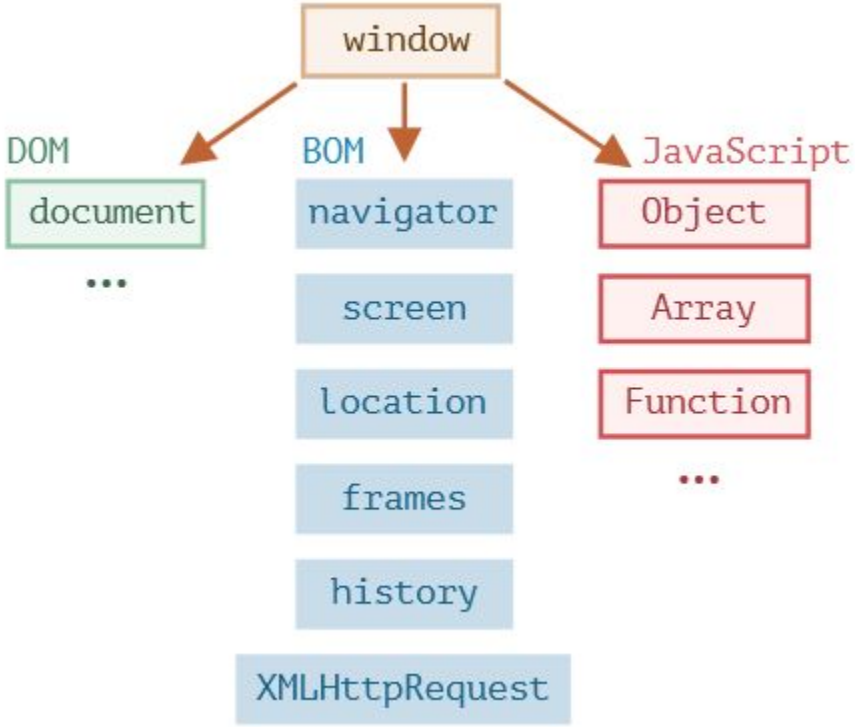
Introdução

Representação e Navegação Básica

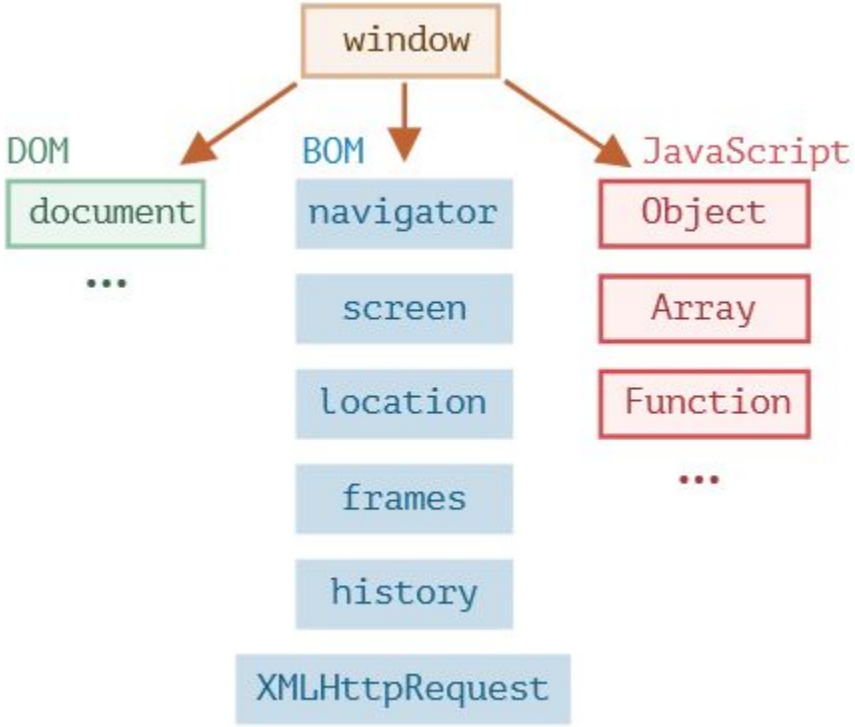
Introdução

- *Javascript* foi inicialmente criada para rodar em *browsers*, porém hoje em dia a linguagem *Javascript* também pode ser utilizada em ambientes diferentes, como servidores e dispositivos que podem executar *Javascript*.
- Cada um desses ambientes é chamado pelo *Javascript* de ambiente *host* (*host environment*).
- Cada ambiente disponibiliza seus próprios objetos e funções em adição ao núcleo da linguagem (*language core*).

Introdução

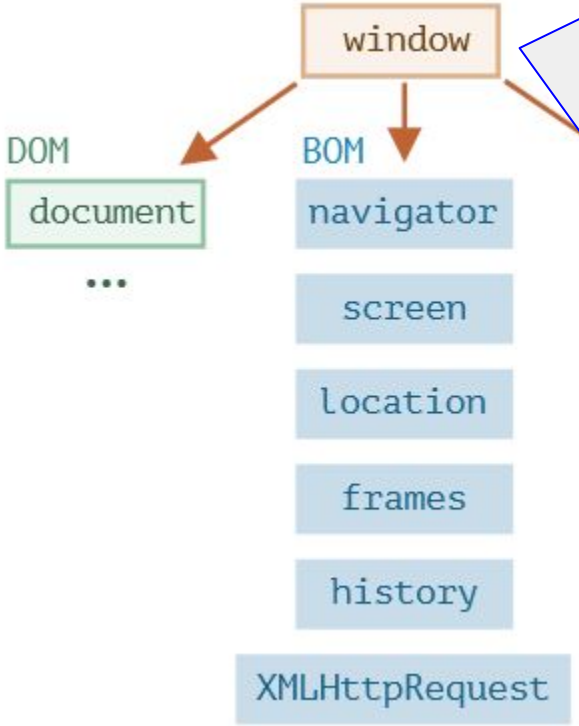

window
DOM
document
...
BOM
navigator
screen
location
frames
history
XMLHttpRequest
JavaScript
Object
Array
Function
...

Introdução



Este era o nosso foco de estudos anteriormente.

Introdução

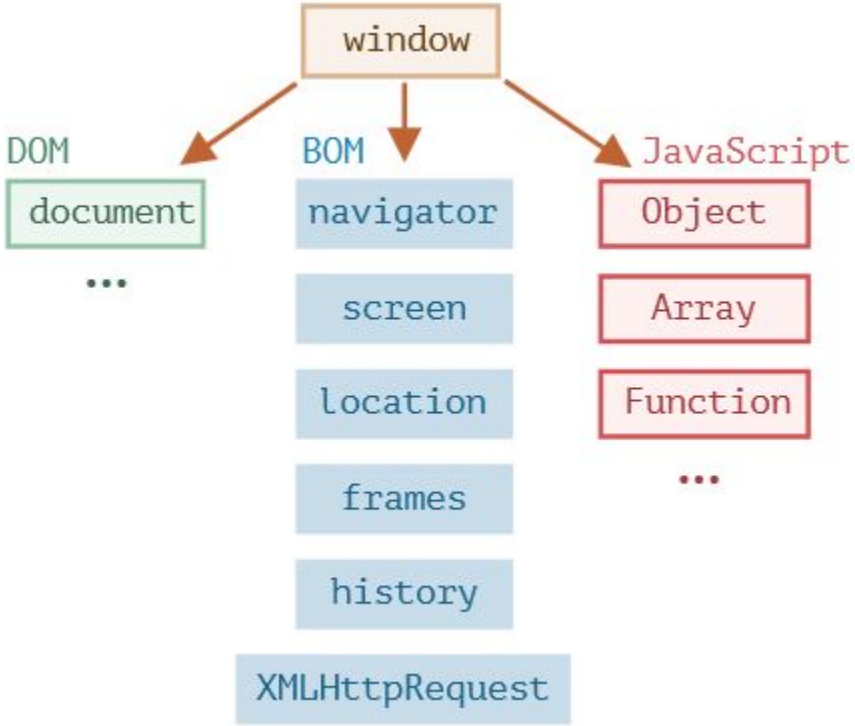


O objeto window é global, representa a janela do browser e disponibiliza métodos para controlar essa janela.
O código abaixo abre a caixa de impressão da janela:

```
window.print();
```

Introdução

Este é o foco de estudos deste material.



DOM

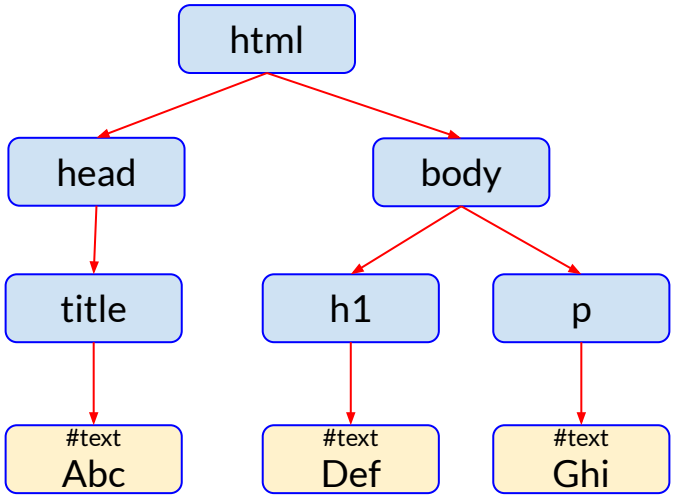
- O *Document Object Model* (DOM) representa o conteúdo de uma página como objetos que podem ser manipulados.
- O objeto document é o ponto de entrada da página, que possibilita alterar ou criar novo conteúdo para a página.
- O código abaixo altera a cor de fundo do body para azul.

```
document.body.style.backgroundColor = 'lightblue';
```

DOM

- Documentos HTML são formados por tags. Cada tag HTML é um objeto no DOM. Tags aninhadas são representadas como objetos filhos do objeto que representa a tag que as contém. O valor dentro de uma tag, mesmo que seja um texto, é um objeto também.

```
<html>
  <head>
    <title>Abc</title>
  </head>
  <body>
    <h1>Def</h1>
    <p>Ghi</p>
  </body>
</html>
```



DOM

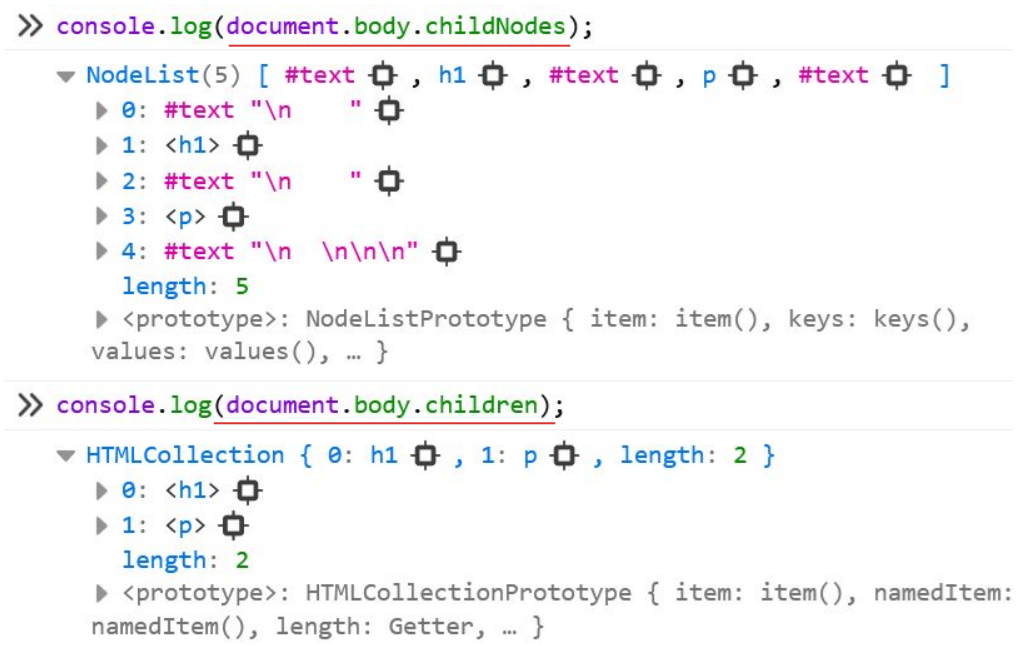
- Existem alguns padrões que são seguidos, tais como:
  - Todo conteúdo após </body> é automaticamente movido para dentro do body.
  - Toda tabela possui o elemento tbody.
    - Mesmo se este elemento tiver sido omitido no HTML, ele existirá no DOM.

DOM

- Para manipularmos um elemento do DOM, primeiramente precisamos alcançar este elemento.
- Esta é a lista contendo os três primeiros elementos:
  - <html> → document.documentElement
  - <head> → document.head
  - <body> → document.body
    - Cuidado: um *script* na *head* poderia retornar *null* para o elemento *body*, pois o mesmo ainda não teria sido carregado.

childNodes e children

- As propriedades childNodes e children retornam os elementos filhos de um objeto.
  - A diferença entre essas duas propriedades é que a propriedade childNodes retorna também os nós texto.



Navegando

- Essas propriedades retornam *collections* e não *arrays*, logo você não conseguirá utilizar os métodos que específicos de arrays nelas.
- Para iterar entre seus elementos, você pode utilizar o seguinte comando:

```
for (let no of document.body.childNodes) {
  console.log(no);
}
```

  - Os nós apresentados por esta iteração são irmãos, pois possuem o mesmo pai, neste caso o elemento *body*.
- Todas as coleções e propriedades de navegação apresentadas até aqui são somente leitura, ou seja, você não pode alterá-las diretamente.

Navegando

- A propriedade *parentNode* de um nó retorna o nó pai.
- A propriedade *nextSibling* retorna o próximo irmão do nó.
- Já a propriedade *previousSibling* retorna o irmão anterior do nó.
  - O que determina o próximo irmão ou o irmão anterior de um nó é a sua posição na *collection* dos nós filhos.
- Se a intenção for navegar sem passar por elementos que são *text nodes*, então as funções possuem “Element” em seu nome:
  - *parentElement*
  - *previousElementSibling*
  - *nextElementSibling*
  - *firstElementChild*
  - *lastElementChild*

Buscando Elementos

getElementById, getElementsBy, querySelectorAll, querySelector*

getElementById

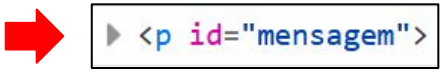
- Se um elemento possuir id, ele pode ser retornado pela função getElementById:

```
<!DOCTYPE HTML>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <script src="testes.js" defer></script>
  <title></title>
</head>
  <body>
    <p id="mensagem" class="xpto">Teste</p>
    <p class="xpto">Teste</p>
    <p class="xyz">Teste</p>
  </body>
</html>
```

getElementById

- Se um elemento possuir id, ele pode ser retornado pela função getElementById:

```
const paragrafo =
document.getElementById('mensagem');
console.log(paragrafo);
```



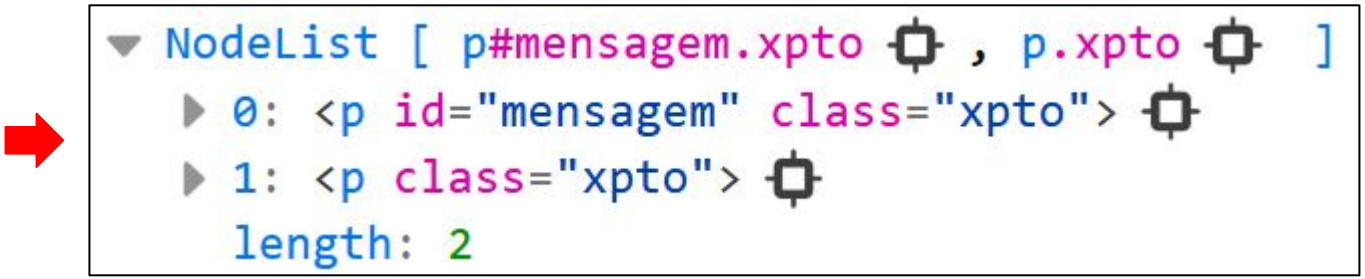
*getElementsBy**

- Existem outras funções que você deve testar:
  - *getElementsByTagName*
  - *getElementsByClassName*
  - *getElementsByName*

querySelectorAll

- A função *querySelectorAll* retorna todos os elementos em acordo com um seletor *CSS*:

```
const elementos = document.querySelectorAll('.xpto');
console.log(elementos);
```



querySelector

- A função *querySelector* retorna o primeiro elemento em acordo com um seletor *CSS*:

```
const elementos = document.querySelector('#mensagem');
console.log(elementos);
```

Tipo de coleção retornada

- O método *querySelectorAll* retorna uma coleção estática.

- Os métodos *getElementsBy** retornam *live collections*, que são automaticamente atualizadas quando o *DOM* é modificado.

Acessando o Conteúdo de um Elemento

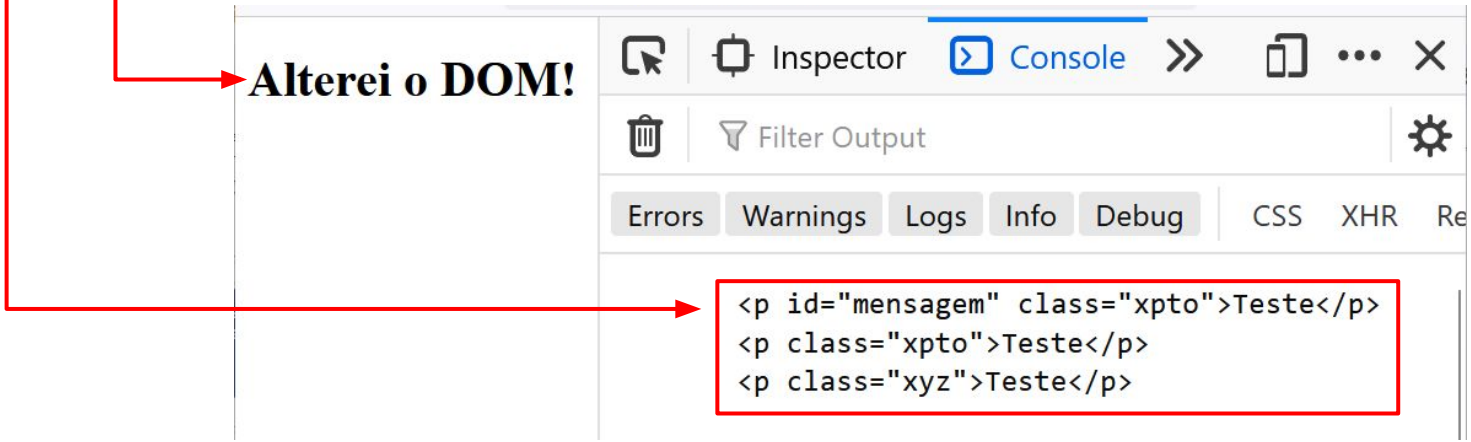
innerHTML, outerHTML e innerText

innerHTML

- A propriedade *innerHTML* retorna o *HTML* dentro de um elemento como uma *string*.
- Essa propriedade também pode ser alterada e esta é uma forma de alterar o *DOM*.
- Alterar esta propriedade causa o recarregamento total dos elementos que estiverem dentro do elemento em que a propriedade foi alterada.

innerHTML

```
console.log(document.body.innerHTML);
document.body.innerHTML = '<h1>Alterei o DOM!</h1>';
```



outerHTML

- A propriedade *outerHTML* retorna o *HTML* do elemento como uma *string*, ou seja, retorna o elemento e seus filhos.

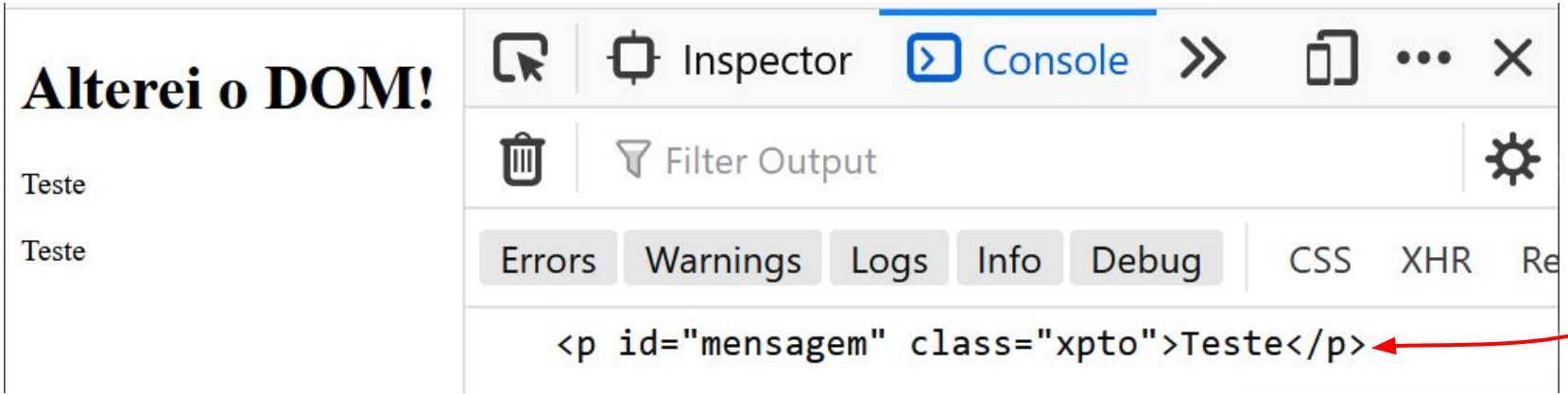
- Essa propriedade também pode ser alterada e esta é uma forma de alterar o *DOM*.

- Alterar esta propriedade causa o recarregamento total do elemento e seu conteúdo.

outerHTML

- Cuidado com referências antigas para o conteúdo:

```
let msg = document.getElementById('mensagem');
msg.outerHTML = '<h1>Alterei o DOM!</h1>';
console.log(msg.outerHTML);
```



Outras Propriedades

textContent, hidden

textContent

- A propriedade textContent retorna todos os textos dos elementos eem as suas tags.

```
console.log(document.body.textContent);
```

hidden

- A propriedade hidden permite ocultar ou desocultar elementos.

```
const msg = document.querySelector('#mensagem');
msg.hidden = true;
```

Atributos e Propriedades

getAttribute, setAttribute, hasAttribute, removeAttribute

Atributos Padrão

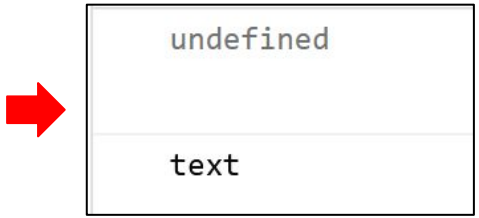
- Existem duas formas para acessarmos os atributos. Primeiro é necessário saber se o atributo é padrão do elemento ou não.

```
<!DOCTYPE HTML>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <script src="testes.js" defer></script>
  <title></title>
</head>
  <body>
    <p id="x" type="abc">Teste</p>
    <input id="y" type="text" />
  </body>
</html>
```

Atributos Padrão

- Apenas elementos que possuem por padrão uma determinada propriedade terão a mesma diretamente representada.

```
var elemX = document.getElementById('x');
var elemY = document.getElementById('y');
console.log(elemX.type);
console.log(elemY.type);
```



Atributos Não-Padrão

- Para acessar atributos, mesmo que estes não sejam padrão do elemento não padrão, utilize os métodos:
  - *getAttribute*
  - *setAttribute*
  - *hasAttribute*
  - *removeAttribute*

Atributos Não-Padrão

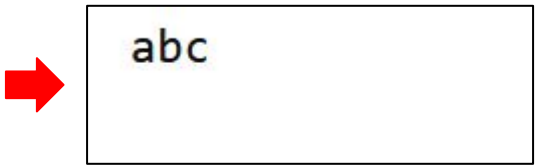
```
var elemX = document.getElementById('x');
var elemY = document.getElementById('y');
console.log(elemX.getAttribute('type'));
console.log(elemY.getAttribute('type'));
```

Atributos Reservados para Programadores

- O uso de atributos não-padrão é útil para quando o programador deseja utilizar este valor em sua codificação, porém existe o risco de uma alteração na linguagem Javascript introduzir um atributo com o mesmo nome, podendo causar problemas.
- Para evitar isso, todo atributo iniciado por “data-” é tem seu uso reservado para programadores.
  - Estes atributos podem ser acessados na propriedade *dataset* do elemento.

Atributos Reservados para Programadores

```
var elemX = document.getElementById('x');
console.log(elemX.dataset.type);
```



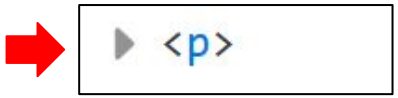
Alterando o Documento

Append, prepend, before, after, replaceWidth

Criando elementos

- O método *createElement* possibilita a criação de um elemento *HTML*. O tipo do elemento (‘div’, ‘p’, ‘input’, etc) deve ser passado por parâmetro para este método.

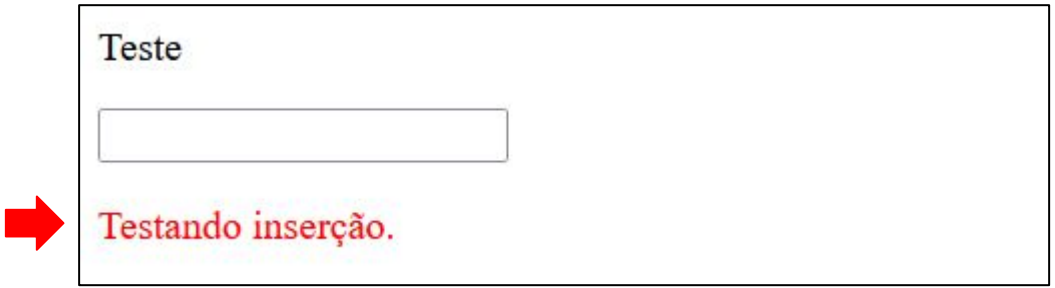
```
let paragrafo = document.createElement('p');
paragrafo.innerHTML = "Testando inserção.";
console.log(paragrafo);
```



Adicionando elementos

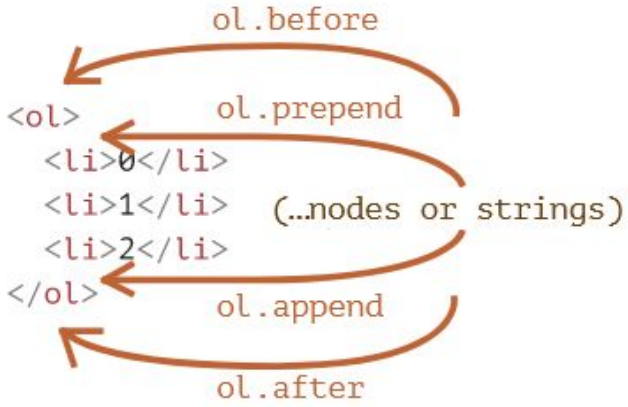
- O método *append* pode ser utilizado para adicionar um elemento ao final dos filhos de outro elemento:

```
let paragrafo = document.createElement('p');
paragrafo.innerHTML = "Testando inserção.";
paragrafo.style.color = 'red';
document.body.append(paragrafo);
```



Adicionando elementos

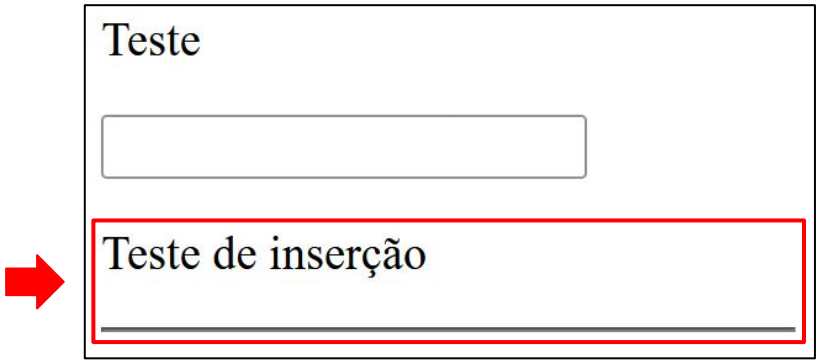
- Existem vários métodos de inserção:
  - no.append:
    - adiciona ao final dos filhos do nó.
  - no.prepend:
    - adiciona no início dos filhos do nó.
  - no.before:
    - adiciona antes do nó.
  - no.after:
    - adiciona após o nó.
  - no.replaceWith:
    - substitui o nó com o valor passado por parâmetro.



Adicionando elementos

- É possível passar por parâmetro mais de um elemento a ser inserido.

```
let p = document.createElement('p');
p.innerHTML = "Teste de inserção";
document.body.append(p, document.createElement('hr'));
```

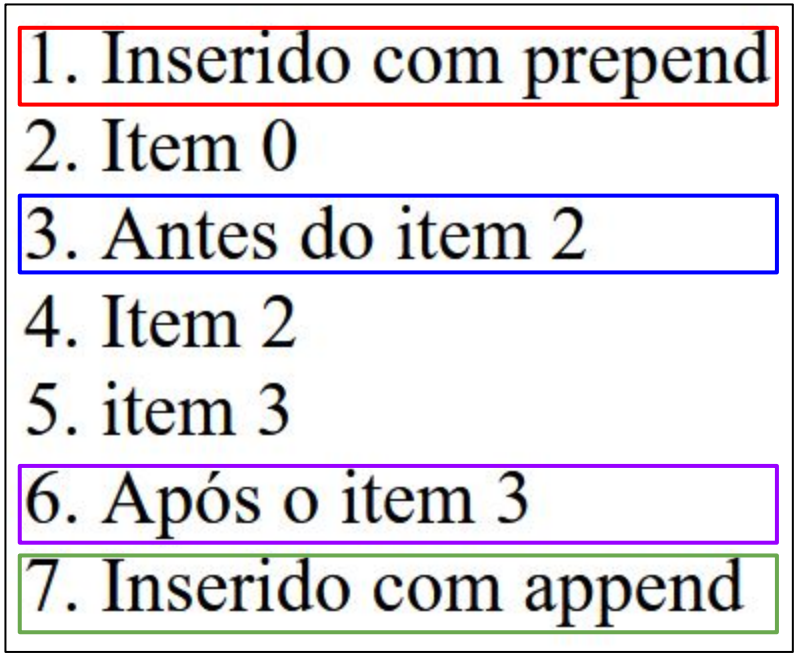


Adicionando elementos

```html
<!DOCTYPE HTML>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <script src="testes.js"
defer></script>
  <title></title>
</head>
  <body>
    <ol id="lista">
      <li id="item0">Item 0</li>
      <li id="item2">Item 2</li>
      <li id="item3">item 3</li>
    </ol>
  </body>
</html>
```

```javascript
let l = document.getElementById('lista');
let i1 = document.getElementById('item0');
let i2 = document.getElementById('item2');
let i3 = document.getElementById('item3');
let b2 = document.createElement('li');
b2.innerHTML = "Antes do item 2";
let a3 = document.createElement('li');
a3.innerHTML = "Após o item 3";
let aOl = document.createElement('li');
aOl.innerHTML = "Inserido com append";
let pOl = document.createElement('li');
pOl.innerHTML = "Inserido com prepend";
i2.before(b2);
i3.after(a3);
l.append(aOl);
l.prepend(pOl);
```

Adicionando elementos




```
let l = document.getElementById('lista');
let i1 = document.getElementById('item0');
let i2 = document.getElementById('item2');
let i3 = document.getElementById('item3');
let b2 = document.createElement('li');
b2.innerHTML = "Antes do item 2";
let a3 = document.createElement('li');
a3.innerHTML = "Após o item 3";
let aOl = document.createElement('li');
aOl.innerHTML = "Inserido com append";
let pOl = document.createElement('li');
pOl.innerHTML = "Inserido com prepend";
i2.before(b2);
i3.after(a3);
l.append(aOl);
l.prepend(pOl);
```

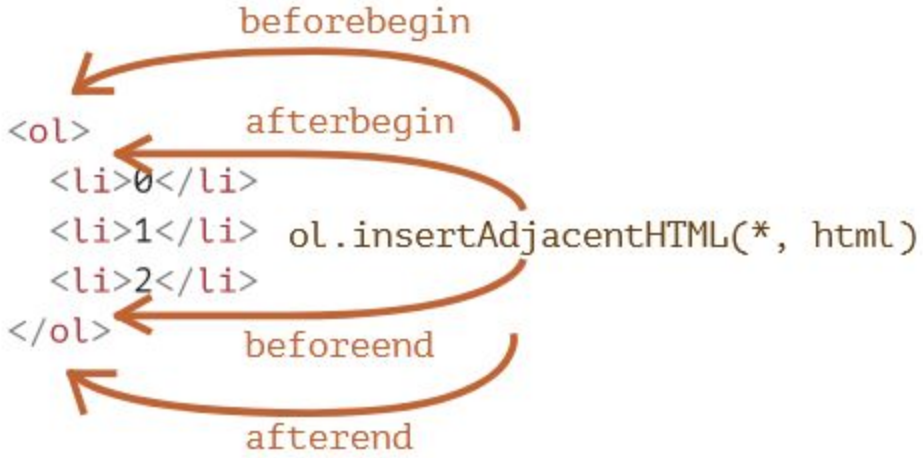
Adicionando HTML

- Os métodos anteriores só permitem a adição de elementos manipulados no *Javascript*. Dessa forma, as *strings* adicionadas são consideradas texto e não *HTML*, mesmo que tenham *tags*.

Adicionando HTML

- Para adicionar *HTML*, é possível utilizar o método:
  - *elem.insertAdjacentHTML(where, html)*
    - *where*: *code word* que indica a posição que o *HTML* será inserido:
      - "beforebegin" – imediatamente antes de elem.
      - "afterend" – imediatamente após o elem.
      - "afterbegin" – dentro do elemento, no início.
      - "beforeend" – dentro do elemento, ao final.
    - *html: string* a ser inserida como *HTML*.

Adicionando HTML



Adicionando HTML

```
const cab = "<h1>Título da Página</h1>";
document.body.insertAdjacentHTML("afterbegin", cab);
```

Adicionando texto e elementos

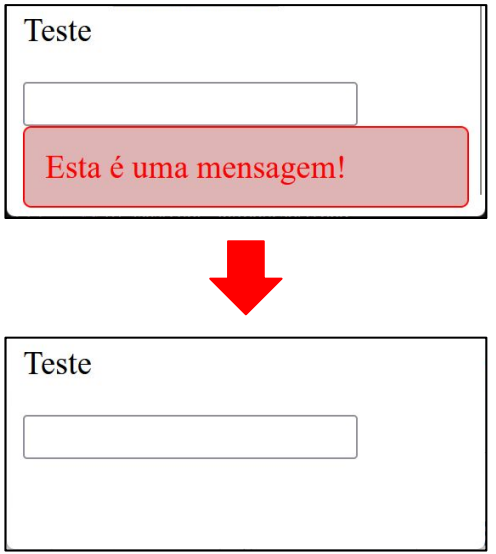
- Para adicionar texto, existe o método *elem.insertAdjacentText(where, text)*
- Para adicionar elemento, existe o método *elem.insertAdjacentElement(where, elem)*
- Ambos os métodos funcionam de forma semelhante ao *insertAdjacentHTML*, exceto pelo tipo de conteúdo que cada um insere.

Removendo elementos

- O método node.remove() remove o nó.

```
let alerta = document.createElement('div');
alerta.innerHTML = `
<div style="border: 1px solid red;color: red;
            background-color: #dfb4b4;
            border-radius: 4px;padding: 10px;"
>
  Esta é uma mensagem!
</div>
`;
document.body.append(alerta);

setTimeout(() => {
  alerta.remove();
}, 2000);
```



Fragmentos

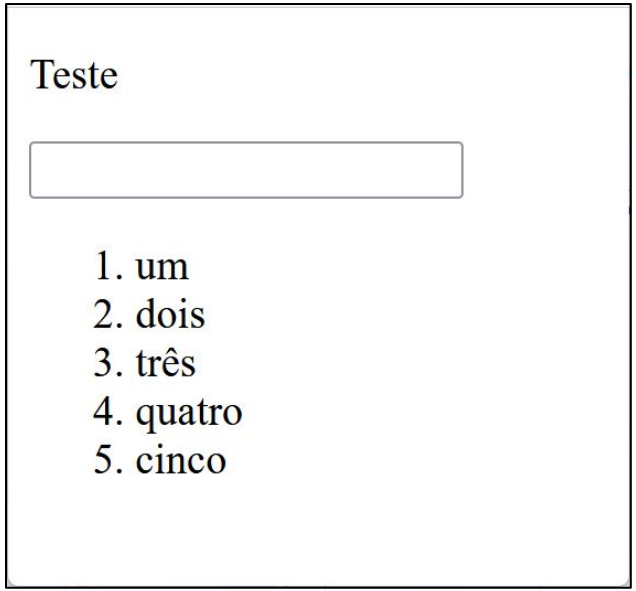
- É possível utilizar DocumentFragment como um fragmento a ser adicionado no HTML:

```
const getItens = () => {
  let fragmento = new DocumentFragment();
  const extenso = ['um','dois','três','quatro','cinco'];
  for(let i=0; i < extenso.length; i++) {
    let li = document.createElement('li');
    li.append(extenso[i]);
    fragmento.append(li);
  }

  return fragmento;
};

let lista = document.createElement('ol');
lista.append(getItens());

document.body.append(lista);
```



Fim de material

Dúvidas?