



# Programação para Internet

---

## Módulo 3

### Páginas Interativas com JavaScript

Prof. Dr. Daniel A. Furtado - FACOM/UFU

Conteúdo protegido por direito autoral, nos termos da Lei nº 9 610/98

A cópia, reprodução ou apropriação deste material, total ou parcialmente, é proibida pelo autor

# Conteúdo da Aula

- Introdução à Linguagem JavaScript
- Recursos Básicos da Linguagem
- Manipulação da Árvore DOM
- JavaScript e Eventos

# O que é JavaScript?

- Linguagem de programação dinâmica
- Utilizada para prover interatividade e dinamismo a websites
- Permite programar o comportamento da página Web na ocorrência de eventos
- Permite alterar o documento HTML por meio da manipulação da árvore DOM
- Comumente referenciada como JS
- Comumente executada no lado cliente, pelo navegador de Internet
  - Linguagem interpretada pelo navegador
  - Não é necessário compilar explicitamente o código JavaScript
- Também pode ser utilizada no lado servidor
  - Utilizando ferramentas como o Node.js
- Não confundir com a linguagem de programação Java

# JavaScript e ECMAScript

- Ecma International – organização que desenvolve padrões
- **ECMAScript** é uma linguagem padronizada, uma especificação
  - ECMA-262 é o nome do padrão propriamente dito
- JavaScript é uma implementação da linguagem ECMAScript
- Outras implementações: JScript e ActionScript
- JavaScript originalmente desenvolvida por Brendan Eich da Netscape

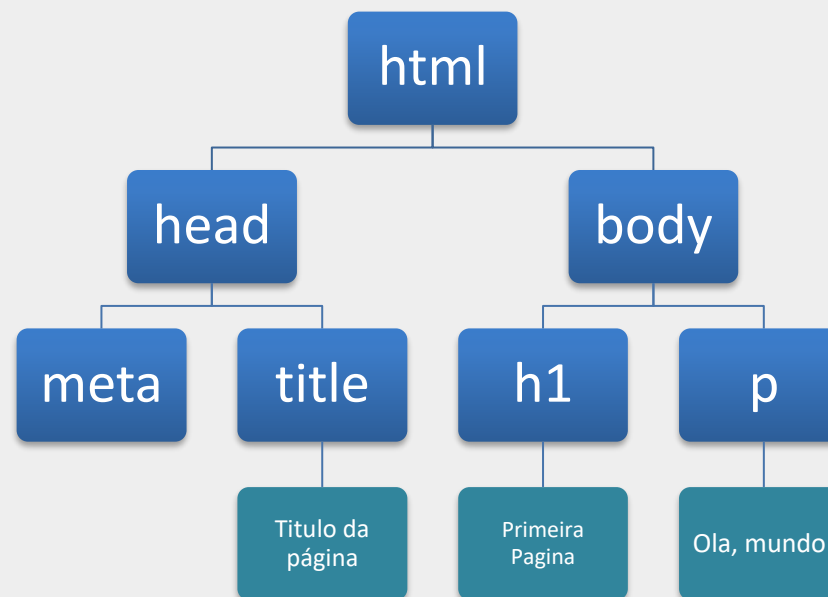
# Manipulação da Árvore DOM

```
<!DOCTYPE html>
<html lang="pt-BR">

  <head>
    <meta charset="UTF-8">
    <title>Titulo da Pagina</title>
  </head>

  <body>
    <h1>Primeira Pagina</h1>
    <p>Ola, mundo!</p>
  </body>

</html>
```



Abstração da Árvore DOM correspondente

**Nota:** Ao carregar uma página, o navegador percorre o respectivo código HTML e monta uma estrutura de dados internamente denominada **árvore DOM**, que é uma representação em memória de toda a estrutura do documento HTML. Nessa estrutura, cada elemento, comentário ou texto do documento HTML é representado como um objeto, denominado **nó**. A estrutura DOM é utilizada para manipular o documento HTML dinamicamente, utilizando programação, com a **DOM API** e a JavaScript.

# Manipulação da Árvore DOM

- Adicionar/modificar o conteúdo aos elementos HTML
- Adicionar novos elementos
- Modificar atributos de elementos
- Modificar estilos CSS
- Ocultar/mostrar elementos
- Remover elementos

A manipulação da árvore DOM (correspondente ao documento HTML) é possível graças a uma Web API denominada DOM API, que pode ser utilizada pelo desenvolvedor por meio da linguagem JavaScript e do navegador de Internet.

# Inserindo Código JavaScript

---

# JavaScript Embutido no HTML

```
<html>

  <head>
    <script>
      // Código JavaScript
    </script>
  </head>

  <body>
    ...
  </body>

</html>
```

*Código JavaScript embutido no  
cabeçalho do documento HTML*

```
<html>

  <head>
    ...
  </head>

  <body>
    ...
    <script>
      // Código JavaScript
    </script>
    ...
  </body>

</html>
```

*Código JavaScript embutido no  
corpo do documento HTML  
(poderia ser depois de </body>)*



# JavaScript em Arquivo Separado

## Arquivo HTML

```
<html>

  <head>

    <script src="arquivoJavaScript.js"></script>

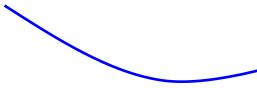
  </head>

  <body>

    ...

  </body>

</html>
```



## Arquivo JS

```
/* arquivoJavaScript.js */

alert('Hello World!');
```

OBS: O conceito de **módulos** em JavaScript não será abordado neste material (inserido com `<script type="module">`)

# Vantagens do JS em Arquivo Separado

- Melhor separação entre conteúdo (HTML) e comportamento (código JS)
- HTML conciso - código mais fácil de ler e manter
- Possibilita reutilizar o código JS em vários arquivos HTML
- Arquivos JavaScript podem ser mantidos em cache pelo navegador
  - Maior agilidade no carregamento

# Execução do Código JavaScript no Navegador

- Fase 1 – Execução durante carregamento – síncrona
  - O código JS inserido com `<script>` é executado durante a fase de carregamento do documento HTML
  - Operações comuns: registro de `event handlers` a serem executadas na 2ª fase
  - Em geral, vários script inseridos com `<script>` são executados na ordem em que aparecem no documento HTML\*
- Fase 2 – Execução em resposta a eventos – assíncrona
  - Código executado na ocorrência de eventos como click de botão, rolagem da página, dado de rede disponível, etc.
  - As funções a serem executadas geralmente são registradas na Fase 1

**\*OBS:** esse comportamento pode ser alterado com os atributos `async` / `defer`

# JavaScript no Navegador - Threading

- JavaScript no navegador executa em modo de **thread única** (single-threaded)
  - Durante a execução do código JavaScript o navegador não responde à interface do usuário (exceto em requisições Ajax)
  - Pode causar uma experiência ruim de navegação ao executar operações que demoram para finalizar
- É possível executar código JavaScript em background (outra thread) usando **web workers**, mas com acesso limitado ao contexto da thread principal

# Recursos da Linguagem

---

# Observações Gerais

- JavaScript é sensível a maiúsculas e minúsculas (case sensitive)
- Declarações podem ou não terminar com o ponto-e-vírgula
- Os tipos das variáveis são definidos automaticamente
- Comentários de linha: `// comentário`
- Comentários de bloco: `/* comentário */`

# Estruturas Condicionais e de Repetição

```
if (expressão) {  
    // operações se verdadeiro  
}  
else {  
    // operações se falso  
}
```

```
switch (expressao) {  
    case condicao1:  
        // operações  
        break;  
  
    case condicaoN:  
        // operações  
        break;  
  
    ...  
    default:  
        // operações  
}
```

```
for (let i = 0; i < 10; i++)  
{  
    // operações  
}
```

```
while (expressao)  
{  
    // operações  
}
```

```
do {  
    // operações  
} while (expressao)
```

# Declaração de Variáveis

**var** nomeDaVariável = valorInicial

- Variável com escopo local se declarada dentro de uma função
- Variável com escopo global se declarada fora de funções
- Pode ser redeclarada e pode ter valor atualizado
- Variáveis globais também podem ser acessadas pelo objeto `window`

**let** nomeDaVariável = valorInicial

- Variável tem escopo restrito ao bloco de código
- Pode ser acessada e atualizada apenas dentro do bloco
- Não pode ser redeclarada no mesmo bloco

**const** nomeDaConstante = valor

- Semelhante a `let`
- Porém não pode ser atualizada
- Deve ser inicializada no momento da declaração



# Exemplo de Variáveis

```
<script>
const pi = 3.14;
var soma = 0;      // soma é uma variável global
for (let i = 1; i <= 10; i++) {
    soma += i;
}

if (soma > 50) {
    let k = soma + pi; // k só pode ser acessada aqui dentro
    var m = k + 1;
    console.log(k);
}

console.log(m); // mostrará o valor de m normalmente
console.log(k); // erro, pois k é restrita ao 'if' acima
</script>
```

# Operadores Aritméticos, Relacionais e Lógicos

## Operadores Aritméticos e Atribuição

| Operador | Significado              |
|----------|--------------------------|
| +        | Adição (e concatenação)  |
| -        | Subtração                |
| *        | Multiplicação            |
| /        | Divisão                  |
| %        | Resto da divisão inteira |
| ++       | Incremento               |
| --       | Decremento               |
| =        | Atribuição               |
| +=       | Atribuição com soma      |
| -=       | Atribuição com sub.      |

## Operadores Relacionais e Lógicos

| Operador | Significado                                      |
|----------|--|
| ==       | Comparação por igualdade                         |
| ===      | Comparação por igualdade, incluindo valor e tipo |
| !=       | Diferente  |
| >        | Maior que  |
| >=       | Maior ou igual a                                 |
| <        | Menor que  |
| <=       | Menor ou igual a                                 |
| &&       | “E” lógico                                       |
|          | “Ou” lógico                                      |
| !        | Negação lógica                                   |

# Operador de Adição e Concatenação

- O operador **+** deve ser utilizado com atenção
- Possibilita **somar** ou **concatenar**, dependendo dos operandos
- Se um dos operandos é uma **string** então será feita a **concatenação**
  - O outro operando é convertido para string, caso não seja
- Se os dois operandos são **numéricos** então é realizada a **soma**
- Exemplos
  - `let x = 5 + 5; // x terá o valor 10`
  - `let y = "5" + 5; // y terá a string '55'`

# Diferença dos Operadores `==` e `===`

## ■ Operador `==`

- Compara apenas valores
- Operandos de tipos diferentes são convertidos e valores comparados

## ■ Operador `===`

- Compara o valor e o tipo dos operandos
- Operandos de tipos diferentes sempre resulta em falso

## ■ Exemplos

- `1 == true;` // retorna true, pois *true* é convertido para 1
- `1 === true;` // retorna false;
- `10 == "10"` // retorna true depois de converter 10 para string
- `10 === "10"` // retorna false;

# Objetos window, navigator, document

## window

- Representa a aba do navegador que contém a página
- Possibilita obter informações ou realizar ações a respeito da janela, como:
  - Obter dimensões: `window.innerWidth` e `window.innerHeight`
  - Executar uma ação quando a aba for carregada, fechada, etc.

## navigator (ou `window.navigator`)

- Representa o navegador de Internet em uso (browser, user-agent)
- Fornece informações como idioma do navegador, geolocalização, memória, etc.
- Exemplo: `alert(navigator.language);` // mostra “pt-BR”

## document (ou `window.document`)

- Representa o documento HTML carregado na aba do navegador
- Possibilita a manipulação da árvore DOM

# Métodos para E/S

- `window.alert`      exibe uma caixa de diálogo para mensagens (botão Ok)
- `window.confirm`      exibe uma caixa de diálogo para confirmação (Ok/Cancelar)
- `window.prompt`      exibe uma caixa de diálogo para entrada de texto
- `document.write`      adiciona conteúdo no documento HTML
- `console.log`      registra conteúdo de `log` no console do navegador
- `console.warn`      registra mensagem de `warning` no console do navegador
- `console.error`      registra mensagem de `erro` no console do navegador

# Declaração de Funções

```
function nomeDaFuncao(par1, par2, par3, ...) {  
    // operações  
    // operações  
    // operações  
}
```

```
function max(a, b) {  
    if (a > b)  
        return a;  
    else  
        return b;  
}
```

```
let maior = max(2, 5);
```

Quando **'return'** não é utilizada, o valor **undefined** é automaticamente retornado

# Tratamento de Eventos

- JavaScript é baseada em eventos
- É possível executar funções na ocorrência de eventos como “clique em botão”, “seleção de item”, “rolagem da página”, etc.
- Funções para tratar eventos podem ser indicadas, na maioria dos casos, de duas formas:
  - Utilizando **propriedades** de eventos;
  - Utilizando o método **addEventListener**



# Tratamento de Eventos

## Propriedades de Tratamentos de Eventos

Permite indicar uma função a ser executada na ocorrência de um evento

```
window.onload = funcaoIniciaPagina;  
// o evento load ocorre quando a página inteira é carregada
```

## Método *addEventListener*

Adiciona uma função a ser executada na ocorrência de um evento

```
window.addEventListener("load", funcaoIniciaPagina);  
// o primeiro parâmetro é o nome do evento e não tem 'on'  
// o segundo parâmetro define a função para tratar o evento,  
// também conhecida como função de callback
```

*addEventListener* tem ainda um 3º parâmetro opcional não apresentado neste exemplo

# Tratamento de Eventos - Exemplo

```
<body>

  <script>

    function mostraMsg() {
      alert('Hello!');
      console.log('Hello!');
    }

    window.onload = mostraMsg;

  </script>

</body>
```

Utilizando a propriedade de evento `onload`

```
<body>

  <script>

    function mostraMsg() {
      alert('Hello!');
      console.log('Hello!');
    }

    window.addEventListener('load', mostraMsg);

  </script>

</body>
```

Utilizando o método `addEventListener`

# Funções Anônimas

```
function funcaoIniciaPagina(event) {  
    // operações  
}  
window.onload = funcaoIniciaPagina;
```

Função tradicional  
definida e depois  
indicada para tratar  
evento

```
window.onload = function (event) {  
    // operações  
}
```

Função anônima  
indicada para tratar  
evento ao mesmo tempo  
em que é definida

# Tratamento de Eventos - Exemplo

```
<body>

  <script>
    window.onload = function() {
      alert('Hello!');
    };
  </script>

</body>
```

```
<body>

  <script>
    window.addEventListener('load', function() {
      alert('Hello!');
    });
  </script>

</body>
```

Exemplo similar ao anterior, porém utilizando funções anônimas

# Eventos load vs DOMContentLoaded

## load

- Ocorre quando a página termina de ser carregada por **completo**
- Só ocorre depois que imagens, arquivos CSS, etc., tenham sido baixados

## DOMContentLoaded

- Ocorre quando o documento é carregado e a árvore DOM termina de ser montada
- Não aguarda pelo carregamento de imagens, arquivos CSS, etc.
- Geralmente ocorre antes do evento **load**

# Exemplo do Evento DOMContentLoaded

```
<body>

  <script>

    document.addEventListener('DOMContentLoaded', function() {

      alert('Hello! Árvore DOM carregada!');
      alert('Agora você pode manipular o documento HTML com a DOM API');

    });

  </script>

</body>
```

# Arrow Function =>

- Define funções sem utilizar a palavra **function**
- Definição abreviada utilizando os caracteres '**=>**'
- Não substitui a definição tradicional em todas as situações

```
window.onload = function () {  
    // operações  
}
```

Indicação de função anônima para tratar evento 'load' no objeto window



```
window.onload = () => {  
    // operações  
}
```

Indicação de função para tratar evento utilizando expressão tipo 'arrow function'

# Arrow Function - Exemplos

- Função com uma única declaração dispensa as chaves

```
window.onload = () => alert('Página carregada...');
```

- Arrow function também pode ter parâmetros

```
window.onload = (e) => alert('Objeto:' + e.target);
```

- Arrow function com um único parâmetro não precisa dos parênteses

```
window.onload = e => alert('Objeto:' + e.target);
```



# Arrays em JavaScript

- Não são tipos de dados primitivos
- São tratados como objetos, com propriedades e métodos
- Podem armazenar valores de tipos diferentes
- Os elementos são acessados por índice numérico
  - Não podem ser acessados por chaves
  - Não são do tipo associativo

# Arrays em JavaScript

- Elementos colocados entre colchetes, separados por vírgula

```
let pares = [2, 4, 6, 8];
```

```
let primeiroPar = pares[0]; // 1º elemento
```

```
let nroElementos = pares.length; // tamanho do vetor
```

- Elementos de diferentes tipos

```
let vetorMisto = [2, 'A', true];
```

- Pode ser iniciado com vazio

```
let pares = [];
```

# Arrays em JavaScript – Outros Métodos

```
let vogais = ['E', 'I', 'O'];
```

```
vogais.push('U')    // adiciona um item no final do vetor
```

```
vogais.pop()        // remove e retorna o último item do vetor
```

```
vogais.shift()       // remove e retorna o primeiro item do vetor
```

```
vogais.unshift('A') // adiciona um item no início do vetor
```

```
vogais.indexOf('E') // retorna a posição da 1ª ocorr. de um item (ou -1)
```

```
vogais.length        // propriedade contendo o número de elem. do vetor
```

# Percorrendo Array com Estrutura *for*

```
let pares = [2, 4, 6, 8];  
for (let i = 0; i < pares.length; i++) {  
    console.log(pares[i]);  
};
```

```
let pares = [2, 4, 6, 8];  
for (let item of pares) {  
    console.log(item);  
};
```

# Percorrendo Array com o Método forEach

```
let pares = [2, 4, 6, 8];
pares.forEach( function (elemento) {
    console.log(elemento);
});

let soma = 0;
pares.forEach( function (elemento) {
    soma += elemento;
});
```

# Percorrendo Array com forEach e Arrow Function

```
let pares = [2, 4, 6, 8];  
let soma = 0;  
  
// soma os elementos  
pares.forEach( elemento => soma += elemento );  
  
// mostra os elementos na janela de console  
pares.forEach( elemento => console.log(elemento) );
```

# Strings

- Definida com aspas simples ou duplas

```
let msg = "JavaScript";
```

- Acessando um caracter

```
let primLetra = msg[0];
```

```
let primLetra = msg.charAt(0);
```

- Contra-barra para caracteres especiais

```
let msg = 'It\'s alright';
```

- Strings com aspas duplas podem conter aspas simples e vice-versa

```
let msg = "It's alright";
```

- Várias propriedades e métodos

```
length, indexOf, substr, split, etc.
```

# Template Literals (ou Template String)

- Strings definidas com o caractere crase (*backtick*): ``minha string``
- Suporta fácil interpolação de variáveis e expressões usando `${ }`
- Maior facilidade para definir strings de múltiplas linhas
- A string pode conter aspas simples ou duplas

```
let a = 1;
let b = 2;
let c = 3;

const delta = b*b - 4*a*c;

console.log(`o discriminante da equacao com
coeficientes ${a}, ${b} e ${c} é ${delta}`);
```



# Objeto Simples (*plain object*, *POJO*)

- Contém apenas dados
- Pode ser definido utilizando chaves { }
- Possui lista de pares do tipo *propriedade* : *valor*
- Criado como instância da classe **Object**

```
let carro = {  
  modelo: "Fusca",  
  ano: 1970,  
  cor: "bege",  
  "motor-hp": 65  
}  
  
console.log(carro.ano);           // 1970  
console.log(carro["motor-hp"]);  // 65
```

# Manipulando a Árvore DOM

---

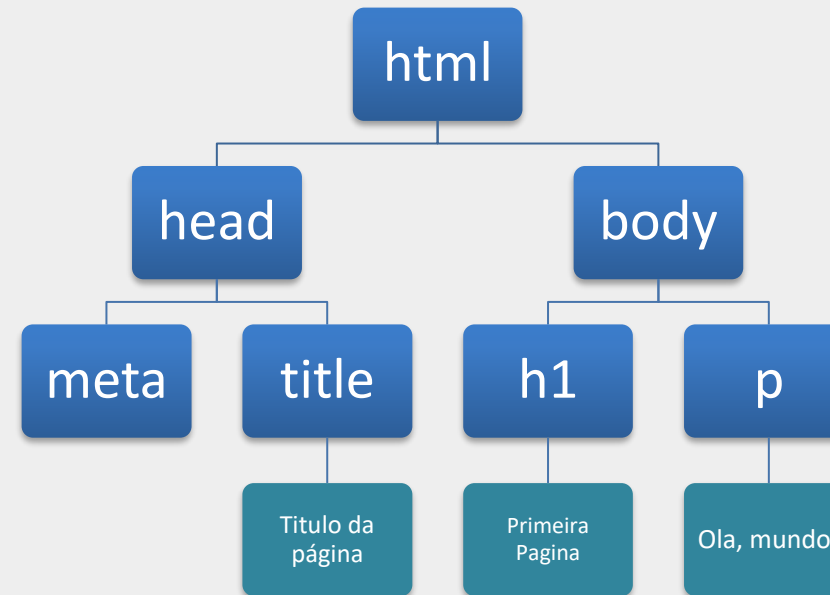
# Document Object Model - DOM

```
<!DOCTYPE html>
<html lang="pt-BR">

  <head>
    <meta charset="UTF-8">
    <title>Titulo da Pagina</title>
  </head>

  <body>
    <h1>Primeira Pagina</h1>
    <p>Ola, mundo!</p>
  </body>

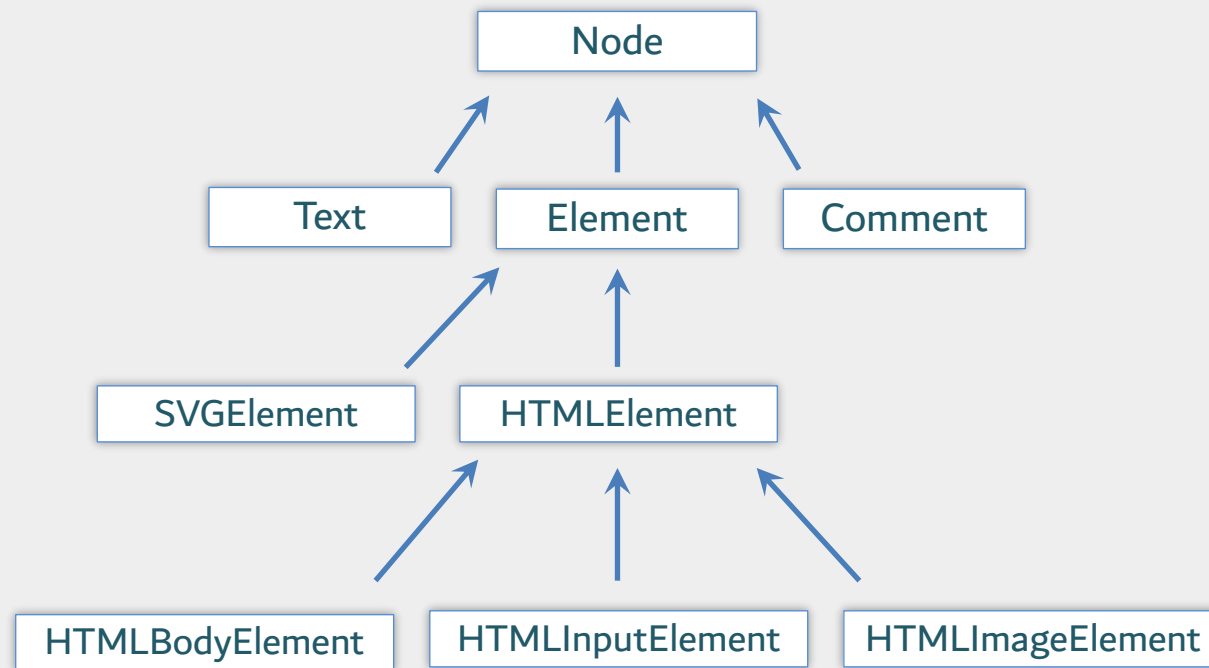
</html>
```



Abstração da Árvore DOM correspondente

**Nota:** Ao carregar uma página, o navegador percorre o respectivo código HTML e monta uma estrutura de dados internamente denominada **árvore DOM**, que é uma representação em memória de toda a estrutura do documento HTML. Nessa estrutura, cada elemento, comentário ou texto do documento HTML é representado como um objeto, denominado **nó**. A estrutura DOM é utilizada para manipular o documento HTML dinamicamente, utilizando programação, com a **DOM API** e a JavaScript.

# Tipos dos Objetos na Árvore DOM



**Node:** classe abstrata com propriedades e métodos utilizados por todos os tipos de nós. Ex.: `no.parentNode`, `no.nextSibling`, `no.childNodes`

**Element:** classe base para os elementos HTML da página. Fornece funcionalidades básicas a nível de elemento. Ex.: `nextElementSibling`, `children`, `getElementById`, `getElementsByTagName`, `querySelector`

**HTMLInputElement:** fornece propriedade e métodos adicionais para elementos 'input', como `value`, `pattern`, `required`, etc.

# Hierarquia de Nós na Estrutura DOM

- Nó **Root**: nó representando o elemento raiz `<html>`
- Nó **Filho**: nó representando um elemento diretamente dentro de outro
- Nó **Pai**: nó representando o elemento que contém o nó filho
- Nós **Irmãos**: nós representando elementos filhos do mesmo pai

# Busca na Árvore DOM

## `document.querySelector`

- Aceita uma string de seleção CSS como parâmetro
- Retorna o **primeiro** nó na árvore DOM (do tipo Element) que atende à seleção
- Ou retorna `null` caso não haja correspondências
- Nenhum elemento é retornado caso o seletor inclua pseudo-elementos

# Busca na Árvore DOM - `document.querySelector`

Retorna o nó correspondente ao primeiro elemento h1 na página

```
const nodeFirstH1 = document.querySelector("h1");
```

Retorna o nó correspondente ao elemento com `id='imagemLogo'`

```
const nodeImgLogo = document.querySelector("#imagemLogo");
```

Retorna o nó correspondente ao primeiro `'li'` filho da primeira `'ul'`

```
const nodeLi = document.querySelector("ul > li");
```

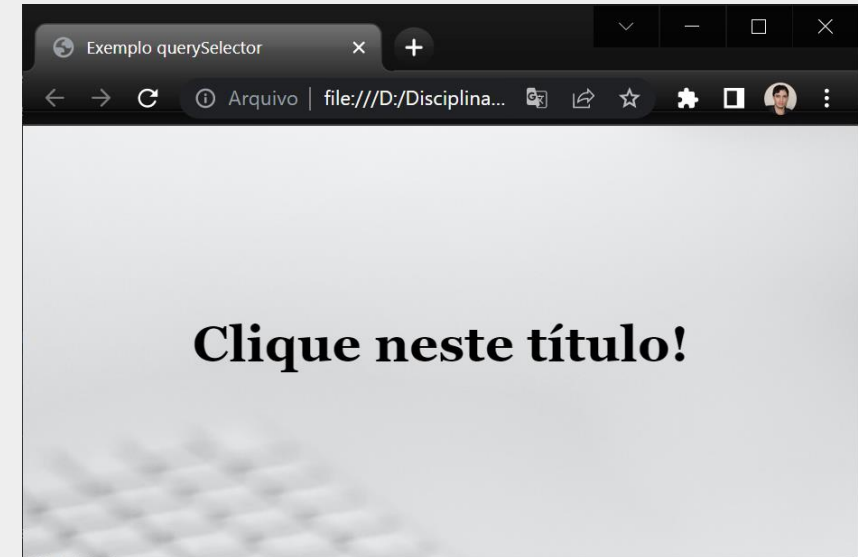
# Busca na Árvore DOM - `document.querySelector`

```
<main>
  <h1>Clique neste título!</h1>
</main>

<script>

  document.addEventListener('DOMContentLoaded', function() {
    const nodeH1 = document.querySelector("h1");
    nodeH1.addEventListener("click", function () {
      nodeH1.textContent = "Você alterou a árvore DOM!";
    });
  });

</script>
```





# Busca na Árvore DOM - `document.querySelector`

```
28 <body>
29
30   <main>
31     
32   </main>
33
34   <script>
35
36     window.addEventListener('load', function() {
37       const imagem = document.querySelector("img");
38       imagem.onclick = function () {
39         this.src = "images/logoFacom.png";
40         this.alt = "Faculdade de Computação";
41       }
42     });
43
44   </script>
45
46 </body>
```



Neste exemplo, a imagem com o logotipo da UFU será alterada quando receber o click do usuário

# Busca na Árvore DOM

## `document.querySelector`

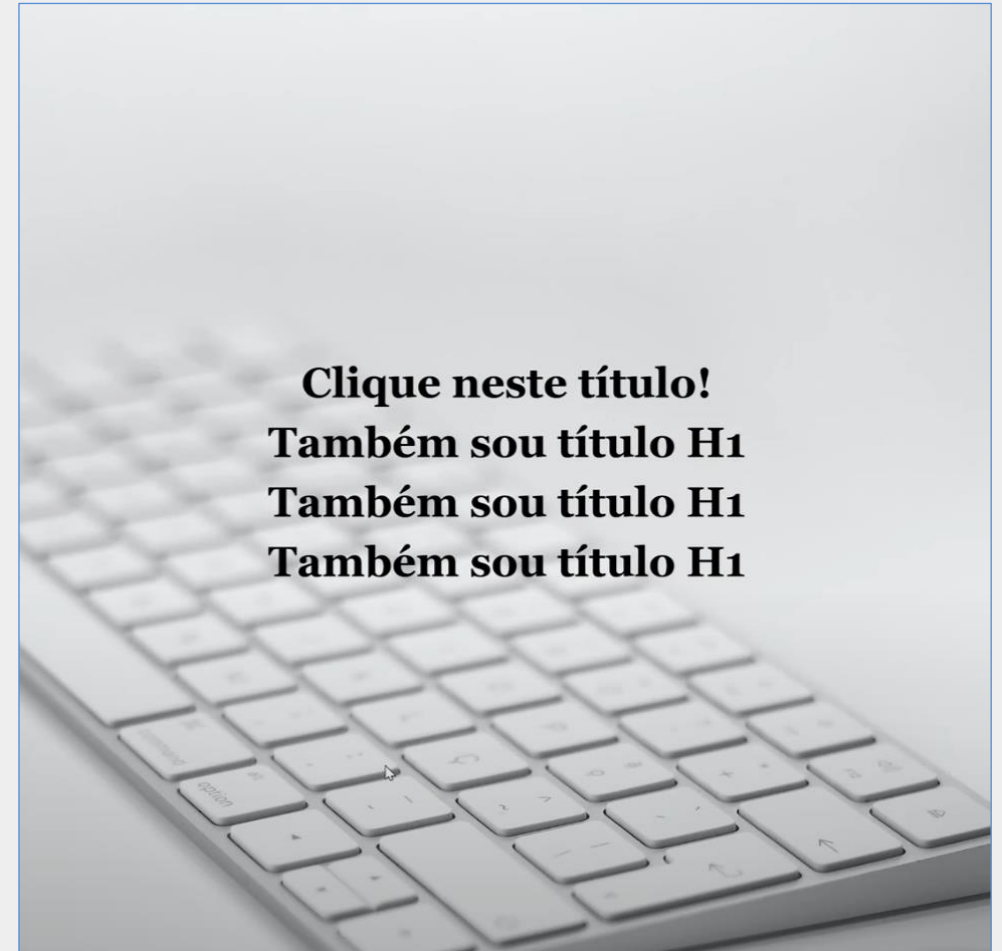
- Aceita uma string de seleção CSS como parâmetro
- Retorna uma lista com **todos** os nós da árvore DOM que atendem à seleção
- Ou retorna `null` caso não haja correspondências

# Busca na Árvore DOM

```
// retorna os nós correspondents a todos os elementos h1 na página
const nodesH1 = document.querySelectorAll("h1");
for (let node of nodesH1) {
    console.log(node.textContent);
}
```

# Manipulação da Árvore DOM - Exemplo

```
32 <main>
33   <h1>Clique neste título!</h1>
34   <h1>Também sou título H1</h1>
35   <h1>Também sou título H1</h1>
36   <h1>Também sou título H1</h1>
37 </main>
38
39 <script>
40
41   document.addEventListener('DOMContentLoaded', function() {
42     const nodeH1 = document.querySelector("h1");
43     nodeH1.addEventListener("click", alteraConteudoDosTitulosH1);
44   });
45
46   function alteraConteudoDosTitulosH1() {
47     const nodesH1 = document.querySelectorAll("h1");
48     for (let node of nodesH1)
49       node.textContent = "Você acabou de alterar a árvore DOM!";
50   }
51
52 </script>
```



Neste exemplo, quando o usuário clicar no **primeiro** título <h1>, **todos** os títulos <h1> terão seu conteúdo alterado para "Você acabou de alterar a árvore DOM!"

# Manipulação da Árvore DOM - Exemplo

```
<main>
  <h1>Clique em algum título!</h1>
  <h1>Clique em algum título!</h1>
  <h1>Clique em algum título!</h1>
  <h1>Clique em algum título!</h1>
</main>

<script>
  document.addEventListener('DOMContentLoaded', function() {
    const nodesH1 = document.querySelectorAll("h1");
    for (let node of nodesH1)
      node.addEventListener("click", () => node.textContent = "Obrigado!");
  });
</script>
```

Neste exemplo, quando o usuário clicar em **qualquer** título <h1>, seu **respectivo** texto será alterado para "Obrigado". Funcionalidade adicionada com *arrow function*.

# Manipulação da Árvore DOM - Exemplo

```
<main>
  <h1>Clique em algum título!</h1>
  <h1>Clique em algum título!</h1>
  <h1>Clique em algum título!</h1>
  <h1>Clique em algum título!</h1>
</main>

<script>
  document.addEventListener('DOMContentLoaded', function() {
    const nodesH1 = document.querySelectorAll("h1");
    for (let node of nodesH1)
      node.addEventListener("click", alteraConteudo);
  });

  function alteraConteudo(e) {
    e.target.textContent = "Obrigado!";
  }
</script>
```

Este exemplo é equivalente ao anterior, porém com a definição de uma função padrão no lugar da *arrow function*.

Repare que a função tem um parâmetro de nome **e**, que receberá o objeto representando o evento.

**e.target** permite acessar o objeto em particular que disparou o evento (título clicado)

# Detalhes do Evento

- Além da propriedade **target** do objeto do evento, há também várias outras propriedades específicas para cada tipo de evento
- Para um evento de **click**, além da propriedade **target**, há ainda outras como:
  - **e.screenX** - coordenada x do clique na tela
  - **e.screenY** - coordenada y do clique na tela
  - **e.ctrlKey** - true ou false indicando se a tecla **ctrl** foi pressionada junto com o click
  - **e.shiftKey** - true ou false indicando se a tecla **shift** foi pressionada com o click
- Para um evento de **teclado**, há outras propriedades como:
  - **e.key** - string correspondente à tecla pressionada (ex.: "Enter", "a", "b", etc.)

# Outras Formas de Realizar Buscas na Árvore DOM

`document.getElementById`

- busca um único elemento utilizando o seu id

`document.getElementsByName`

- busca os elementos pelo valor do atributo **name** do elemento

`document.getElementsByTagName`

- busca os elementos pelo nome da tag HTML, como `img`, `h1`, etc.

`document.getElementsByClassName`

- busca os elementos pelo valor do atributo `class`



# Exemplo de *document.getElementsByName*

```
...  
<input type="radio" name="estadoCivil" value="solteiro">  
<input type="radio" name="estadoCivil" value="casado">  
<input type="radio" name="estadoCivil" value="divorciado">  
...  
<script>  
    const radiosEstCiv = document.getElementsByName("estadoCivil");  
    for (let radio of radiosEstCiv) {  
        if (radio.checked)  
            alert(radio.value);  
    }  
</script>
```

## Exemplo de *document.getElementsByTagName*

```
...  
<input type="radio" name="estadoCivil" value="solteiro">  
<input type="radio" name="estadoCivil" value="casado">  
<input type="text" name="bairro">  
<input type="text" name="cidade">  
...  
<script>  
    const listaDeInputs = document.getElementsByTagName("input");  
    for (let input of listaDeInputs) {  
        alert(input.name);  
    }  
</script>
```

Busca correspondente utilizando *querySelectorAll*

```
document.querySelectorAll("input");
```

## Exemplo de *document.getElementsByClassName*

```
...  
<ul class="nav"> ... </ul>  
...  
<script>  
    const primListaNav = document.getElementsByClassName("nav")[0];  
</script>
```

Busca correspondente utilizando *querySelector*

```
document.querySelector(".nav");
```

# Exercício 1

- Crie uma função JavaScript de nome `showMessage('minha mensagem')` para exibir uma mensagem em um pequeno painel no centro da página
- Construa o painel (caixa de mensagem) utilizando um elemento `<div>`
- Utilize CSS para definir o estilo do painel e deixá-lo inicialmente oculto. Utilize o posicionamento absoluto para posicioná-lo centralizado na vertical e na horizontal
- A função `showMessage` deve alterar o **conteúdo** do `<div>` por meio da manipulação do respectivo objeto na árvore DOM. Utilize a propriedade `textContent` do objeto. Para apresentar a caixa de mensagem, altere sua propriedade **visibility** com JavaScript (`objetoDiv.style.visibility = 'visible'`)
- Acrescente um botão na página. Quando ele for acionado, uma mensagem qualquer deve ser apresentada utilizando a caixa de mensagem criada.

# Acesso ao Conteúdo dos Elementos

- Propriedade `textContent`
- Propriedade `innerText`
- Propriedade `innerHTML`

# Acesso ao Conteúdo dos Elementos

## Propriedade `textContent`

- Se o conteúdo do elemento é textual, retorna esse texto
- Se o elemento possui filhos, retorna a concatenação do `textContent` dos filhos
- Uma alteração do valor removerá todos os nós filhos e substituirá pelo novo texto

## Propriedade `innerText`

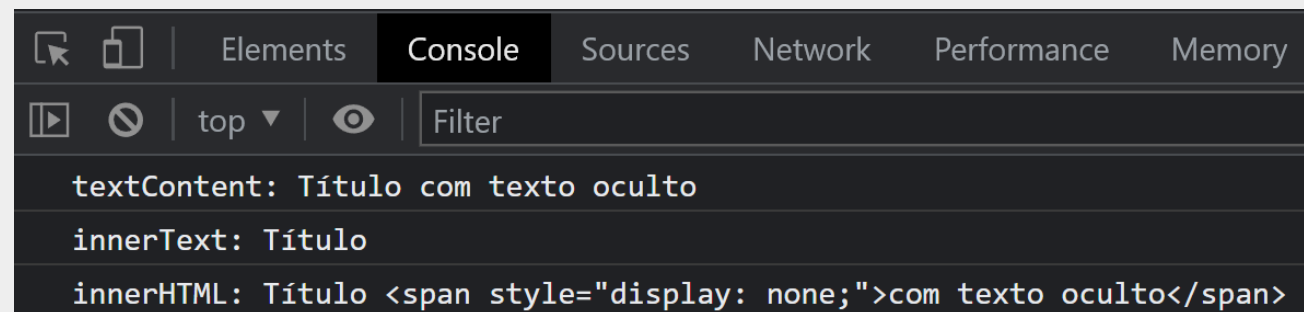
- Semelhante a `textContent`, porém **não inclui** conteúdos de elementos que “não podem ser lidos” pelo usuário, como conteúdo oculto com CSS, conteúdo de tags como `<script>`, `<style>`, etc.

## Propriedade `innerHTML`

- Retorna o conteúdo do elemento e de seus descendentes, incluindo as tags HTML
- Quando alterada, o novo conteúdo é avaliado pelo navegador e pode resultar na criação de nós descendentes na estrutura DOM
- **OBS:** possibilidade de ataques XSS e desempenho inferior a `textContent`.

# textContent vs innerText vs innerHTML

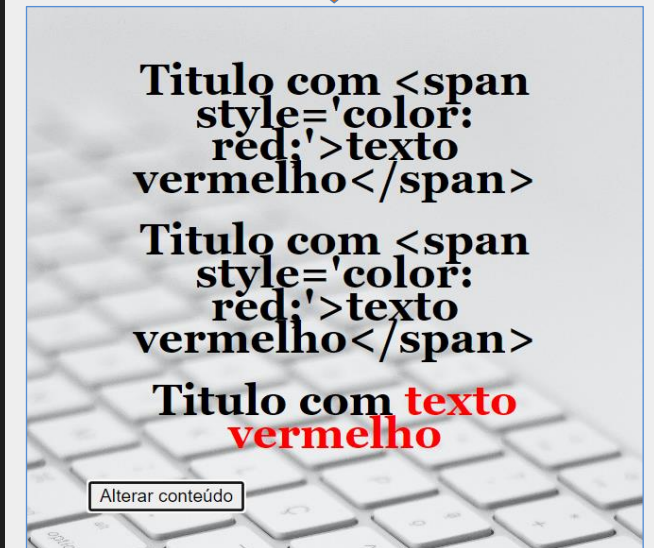
```
30 <body>
31
32   <main>
33     <h1>Título
34     |   <span style="display: none;">com texto oculto</span>
35     </h1>
36   </main>
37
38   <script>
39
40     document.addEventListener('DOMContentLoaded', function () {
41       const nodeH1 = document.querySelector("h1");
42       console.log("textContent: " + nodeH1.textContent);
43       console.log("innerText: " + nodeH1.innerText);
44       console.log("innerHTML: " + nodeH1.innerHTML);
45     });
46
47   </script>
48
49 </body>
```



# textContent vs innerText vs innerHTML

```
<main>
  <h1>Título</h1>
  <h1>Título</h1>
  <h1>Título</h1>
  <button>Alterar conteúdo</button>
</main>

<script>
  document.addEventListener('DOMContentLoaded', function () {
    const botao = document.querySelector("button");
    botao.onclick = function () {
      const nodesH1 = document.querySelectorAll("h1");
      const novoConteudo =
        "Título com <span style='color: red;'>texto vermelho</span>";
      nodesH1[0].textContent = novoConteudo;
      nodesH1[1].innerText = novoConteudo;
      nodesH1[2].innerHTML = novoConteudo;
    }
  });
</script>
```





# textContent vs innerText vs innerHTML

```
<main>
  <h1>Título</h1>
  <h1>Título</h1>
  <h1>Título</h1>

  <input type="text">
  <button>Alterar conteúdo</button>
</main>

<script>
  document.addEventListener('DOMContentLoaded', function () {
    const botao = document.querySelector("button");
    botao.onclick = function () {
      const nodesH1 = document.querySelectorAll("h1");
      const novoConteudo = document.querySelector("input").value;
      nodesH1[0].textContent = novoConteudo;
      nodesH1[1].innerText = novoConteudo;
      nodesH1[2].innerHTML = novoConteudo;
    }
  })
</script>
```

Título  
Título  
Título



Alterar conteúdo



Essa página diz  
ahaaa

OK

**Ataque XSS:** o código JavaScript do usuário será executado devido ao uso inadequado da propriedade **innerHTML**

# Alterando Estilos CSS de forma *inline*

- Utiliza-se a propriedade **style** do objeto
- Neste caso, a alteração ocorre com CSS *inline*
- Nomes das propriedades segue padrão *CamelCase*

## CSS

color

font-family

background-color

## JavaScript

node.**style**.color

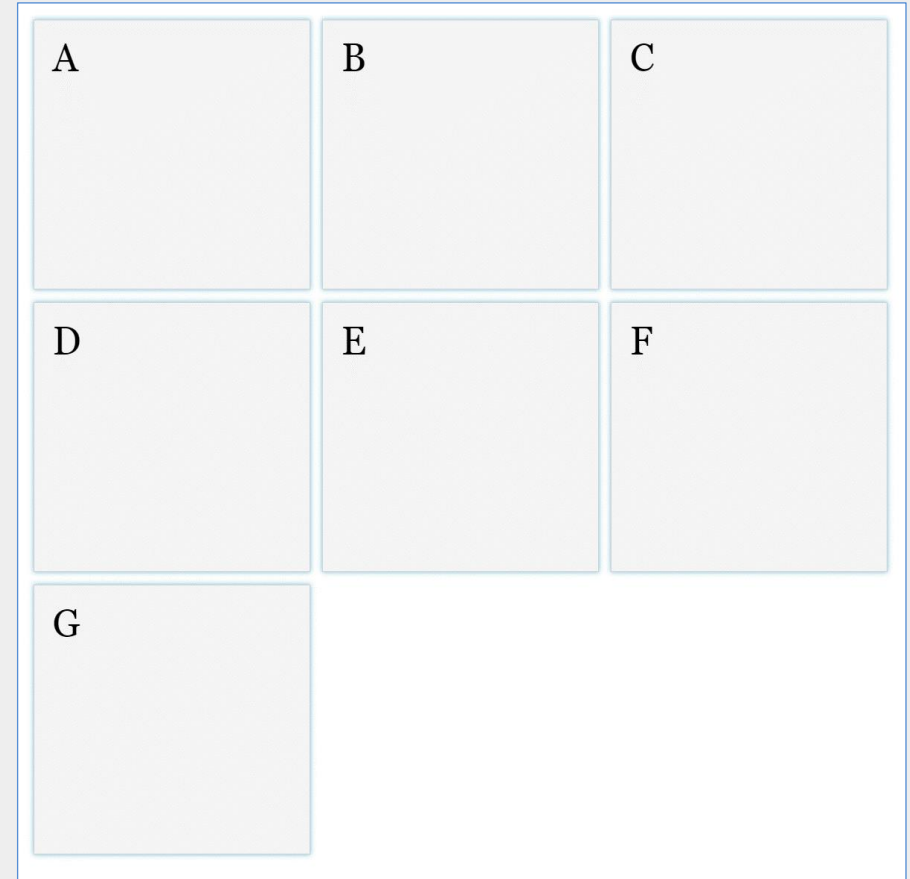
node.**style**.fontFamily

node.**style**.backgroundColor

# Alterando Estilos CSS de forma *inline*

```
<main>
  <article>A</article>
  <article>B</article>
  <article>C</article>
  <article>D</article>
  <article>E</article>
  <article>F</article>
  <article>G</article>
</main>

<script>
  window.onload = function () {
    const artigos = document.querySelectorAll("article");
    for (let artigo of artigos) {
      artigo.onclick = () => artigo.style.visibility = 'hidden';
    }
  }
</script>
```



Neste exemplo, o clique em um artigo (caixa cinza) fará com que o mesmo fique oculto.

# Manipulando Atributos

Para a maioria dos atributos dos elementos HTML da página há uma propriedade de **mesmo nome** no objeto correspondente da árvore DOM

```
...  
<input type="text" id="aabb" name="ccdd" value="rua abc">  
...  
<script>  
    const campoRua = document.querySelector("input");  
    console.log( campoRua.id );      // mostra 'aabb'  
    console.log( campoRua.name );    // mostra 'ccdd'  
    console.log( campoRua.value );   // mostra 'rua abc'  
    campoRua.name = "novo valor";    // alt. o val. do atrib. 'name'  
</script>
```

# Manipulando Atributos

Alguns atributos são acessados de forma diferenciada

---

Atributo HTML

JavaScript

class

`node.className`

data-matricula

`node.dataset.matricula`

data-num-matricula

`node.dataset.numMatricula`

`node.dataset["numMatricula"]`

# Manipulando Atributos - Exemplo

```
28 <body>
29
30   <main>
31     
32   </main>
33
34   <script>
35
36     window.addEventListener('load', function() {
37       const imagem = document.querySelector("img");
38       imagem.onclick = function () {
39         this.src = "images/logoFacom.png";
40         this.alt = "Faculdade de Computação";
41       }
42     });
43
44   </script>
45
46 </body>
```



Neste exemplo, a imagem com o logotipo da UFU será alterada quando receber o **click** do usuário

# Manipulando Atributos

## `node.getAttribute`

- Permite acessar o valor do atributo conforme aparece na HTML (string)
- Em alguns casos retorna um valor igual à respectiva propriedade
- Há casos em que retorna um valor diferente da propriedade
- Atributos não padronizados **devem** ser acessados com `getAttribute`
  - Propriedades não são criadas para atributos não padronizados

# Manipulando Atributos

## node.getAttribute

```
...  
<input type="radio" checked>  
...  
<script>  
    const campo = document.querySelector("input");  
    const a = campo.checked;           // retorna true  
    const b = campo.getAttribute("checked"); // retorna ""  
</script>
```



# Manipulando Atributos

## node.getAttribute

```
...  
<h1 style="color: blue">Título Qualquer</h1>  
...  
<script>  
    const titulo = document.querySelector("h1");  
    alert(titulo.style);           // Mostra [object CSSStyleDeclaration]  
    alert(titulo.style.color);     // Mostra blue  
    alert(titulo.getAttribute("style")); // Mostra 'color: blue'  
</script>
```

# Manipulando Atributos

## `node.setAttribute`

- Define o valor de um atributo
- Se o atributo existe, atualiza o valor
- Caso contrário, cria um novo atributo com o respectivo valor

# Manipulando Atributos

## node.setAttribute

```
...  
<h1 id="tituloTeste1">Título Qualquer</h1>  
...  
<script>  
    const titulo = document.querySelector("h1");  
    titulo.setAttribute("id", "novoIdDoTitulo");  
</script>
```

# Manipulação da Árvore DOM

## `node.firstChild`

- retorna o primeiro nó filho do elemento
- pode incluir nó de texto ou nó de comentário

## `node.firstChildElement`

- retorna o primeiro nó filho do tipo elemento

## `node.lastChild`

- retorna o último nó filho
- pode incluir nó de texto ou nó de comentário

## `node.lastElementChild`

- retorna o último nó filho do tipo elemento

# Manipulação da Árvore DOM

## `node.hasChildNodes`

- retorna verdadeiro caso o nó tenha filhos

## `node.childNodes`

- retorna uma lista com **todos** os nós filhos
- inclui nós de **texto**, nós de **comentário** e nós do tipo **elemento**

## `node.children`

- retorna lista contendo apenas nós filhos do **tipo elemento**

# Manipulação da Árvore DOM

`node.appendChild(novoNo)`

- acrescenta um nó filho no final da lista de filhos

`node.removeChild(noFilhoASerRemovido)`

- remove um nó filho (parâmetro) da lista de filhos

`node.remove()`

- remove o **próprio nó** da lista de filhos do nó pai

`node.parentNode`

- retorna o nó pai do nó em questão

# Manipulação da Árvore DOM

## `node.nextSibling`

- retorna o próximo nó irmão (nó de **qualquer tipo**)

## `node.previousSibling`

- retorna o nó irmão anterior (nó de **qualquer tipo**)

## `node.nextElementSibling`

- retorna o próximo nó irmão do **tipo elemento**

## `node.previousElementSibling`

- retorna o nó irmão anterior do **tipo elemento**

# Manipulação da Árvore DOM

`document.createElement("elementoASerCriado")`

- cria um novo nó do tipo **Element**
- Ex.: `let novoSpan = document.createElement("span");`

`node.cloneNode(deep)`

- duplica o objeto correspondente ao nó
- se o parâmetro `deep` for **true**, clona também os nós filhos
- pode ser usado para duplicar uma parte do documento HTML (o clone precisa ser inserido na árvore DOM)



# Manipulação da Árvore DOM - Exemplo

```
20 <body>
21   <div>
22     <label for="interesse">Interesse:</label>
23     <input type="text" id="interesse" name="interesse">
24     <button>Adicionar</button>
25   </div>
26   <ol>
27     <li>Shows</li>
28     <li>Filmes</li>
29   </ol>
30
31   <script>
32
33     window.onload = function () {
34       const botaoAdicionar = document.querySelector("button");
35       botaoAdicionar.addEventListener("click", adicionaInteresse);
36     }
37
38     function adicionaInteresse() {
39       const campoInteresse = document.querySelector("#interesse");
40       const listaInteresses = document.querySelector("ol");
41
42       const novoLi = document.createElement("li");
43       novoLi.textContent = campoInteresse.value;
44       listaInteresses.appendChild(novoLi);
45       campoInteresse.value = '';
46     }
47
48   </script>
49 </body>
```

Interesse:

1. Shows
2. Filmes
3. Esportes

# Manipulação da Árvore DOM - Exemplo

```
62 window.onload = function () {
63   const campoInteresse = document.querySelector("#interesse");
64   campoInteresse.addEventListener("keyup", e => {
65     if (e.key === "Enter")
66       adicionaInteresse();
67   });
68 }
69
70 function adicionaInteresse() {
71
72   const campoInteresse = document.querySelector("#interesse");
73   const listaInteresses = document.querySelector("ol");
74
75   const novoLi = document.createElement("li");
76   const novoSpan = document.createElement("span");
77   const novoBotao = document.createElement("button");
78
79   novoSpan.textContent = campoInteresse.value;
80   novoBotao.textContent = 'X';
81
82   novoLi.appendChild(novoSpan);
83   novoLi.appendChild(novoBotao);
84   listaInteresses.appendChild(novoLi);
85
86   novoBotao.onclick = function () {
87     listaInteresses.removeChild(novoLi);
88     // ou: novoLi.remove();
89     // ou: novoLi.parentNode.removeChild(novoLi);
90   }
91
92   campoInteresse.value = '';
93 }
```

Interesse:

1. Ciência ×
2. Esportes ×
3. Viagens ×
4. Filmes ×

# Outras Propriedades do Objeto *document*

- `document.head` - acesso direto ao nó corresp. ao elemento <head>
- `document.body` - acesso direto ao nó corresp. ao elemento <body>
- `document.title` - acesso direto ao nó corresp. ao elemento <title>
- `document.location` - objeto com URL da página. Pode ser modificado.
- `document.forms` - retorna coleção de todos os formulários (<form>)
- `document.images` - retorna coleção de todas as imagens (<img>)
- `document.anchors` - retorna coleção de todos os links (<a>)

# Exemplos de Uso de *document.forms*

```
<form name="cadastro">  
  Produto: <input name="produto">  
  Último Nome: <input name="ultimo-nome">  
</form>
```

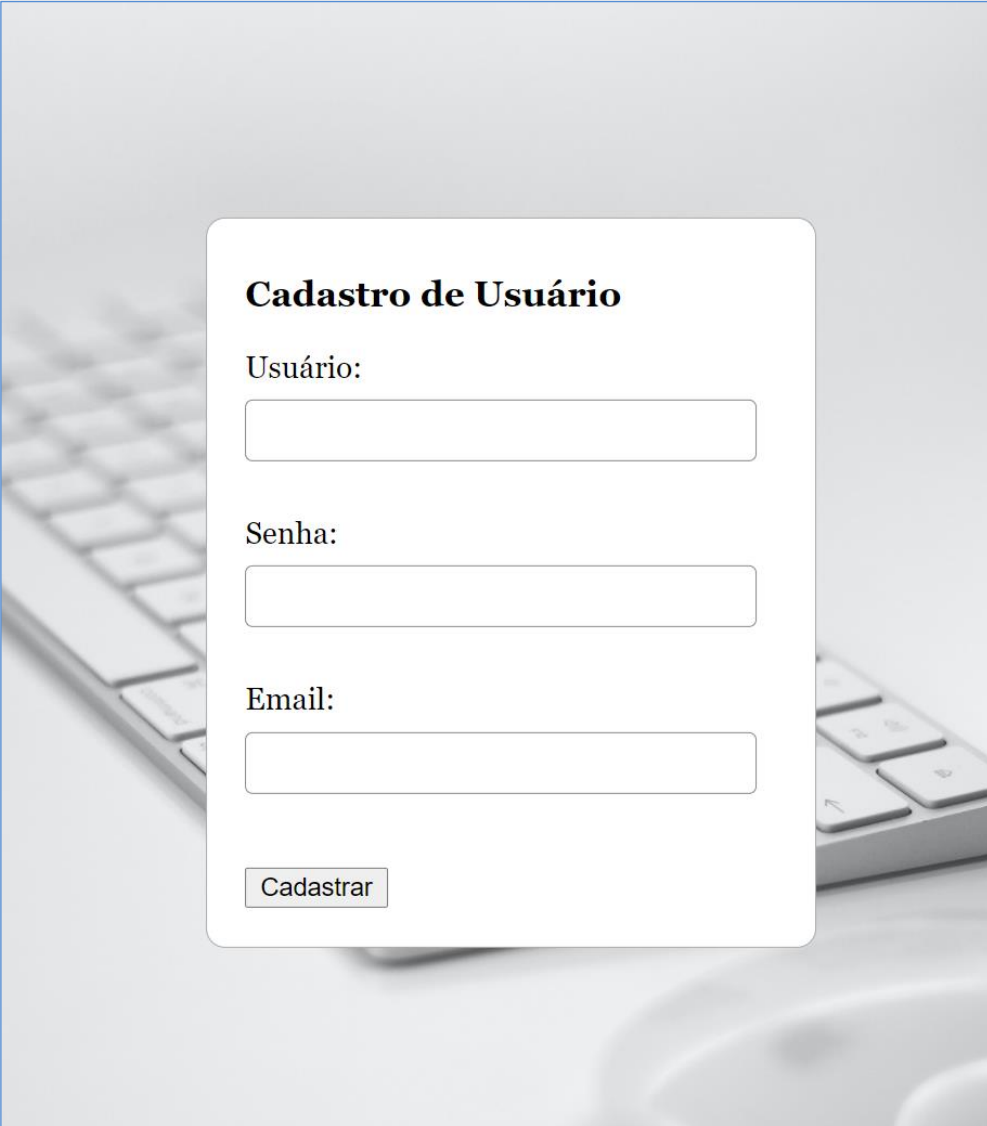
```
const campoProduto = document.forms.cadastro.produto;  
const valorDoCampo = campoProduto.value;  
const ultNome = document.forms.cadastro["ultimo-nome"].value;
```

## Outras Formas

- `const campoProduto = document.forms.cadastro.elements.produto;`
- `const campoProduto = document.forms["cadastro"].elements.produto;`
- `const campoProduto = document.forms[0]["ultimo-nome"];`
- `const campoProduto = document.forms["cadastro"]["produto"];`
- `const campoProduto = document.forms.item(0)["produto"];`
- `const campoProduto = document.forms.namedItem("cadastro")["produto"];`

# Validação de Formulário - Exemplo Simples

```
61 <form name="formCadastro" action="action.php">
62
63   <div>
64     <label for="usuario">Usuário:</label>
65     <input type="text" id="usuario" name="usuario">
66     <span></span>
67   </div>
68
69   <div>
70     <label for="senha">Senha:</label>
71     <input type="password" id="senha" name="senha">
72     <span></span>
73   </div>
74
75   <div>
76     <label for="email">Email:</label>
77     <input type="email" id="email" name="email">
78     <span></span>
79   </div>
80
81   <button>Cadastrar</button>
82
83 </form>
```

A visual representation of the HTML form code shown on the left. It features a white rounded rectangle centered on a blurred background of a laptop keyboard. The rectangle has a title 'Cadastro de Usuário' in bold. Below the title are three labels: 'Usuário:', 'Senha:', and 'Email:'. Each label is followed by a text input field. The 'Senha:' field is a password input. At the bottom of the rectangle is a button labeled 'Cadastrar'.

# Validação de Formulário - Exemplo Simples

```
window.onload = function () {  
  document.forms.formCadastro.onsubmit = validaForm;  
}  
  
function validaForm (e) {  
  let form = e.target;  
  let formValido = true;  
  
  const spanUsuario = form.usuario.nextElementSibling;  
  const spanSenha = form.senha.nextElementSibling;  
  const spanEmail = form.email.nextElementSibling;  
  
  spanUsuario.textContent = "";  
  spanSenha.textContent = "";  
  spanEmail.textContent = "";  
  
  if (form.usuario.value === "") {  
    spanUsuario.textContent = 'O usuário deve ser preenchido';  
    formValido = false;  
  }  
  if (form.senha.value === "") {  
    spanSenha.textContent = 'A senha deve ser preenchida';  
    formValido = false;  
  }  
  if (form.email.value === "") {  
    spanEmail.textContent = 'O email deve ser preenchido';  
    formValido = false;  
  }  
  
  if (! formValido)  
    e.preventDefault();  
}
```

O método `preventDefault` impede a execução da ação padrão associada ao evento. Neste caso, a chamada impede que o formulário seja submetido.

## Cadastro de Usuário

Usuário:

O usuário deve ser preenchido

Senha:

A senha deve ser preenchida

Email:

O email deve ser preenchido

Cadastrar

# Referências

- <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- <https://www.ecma-international.org/ecma-262/>
- David Flanagan. **JavaScript: The Definitive Guide**. 7ª ed., 2020.
- Jon Duckett. **JavaScript and JQuery: Interactive Front-End Web Development**.