



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

1. Introdução à Linguagem JavaScript
2. Recursos Básicos da Linguagem
3. Manipulação da Árvore DOM
4. JavaScript e Eventos

O que é JavaScript?

- Linguagem de programação dinâmica
- Comumente 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

JavaScript Permite Manipular a Árvore DOM

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

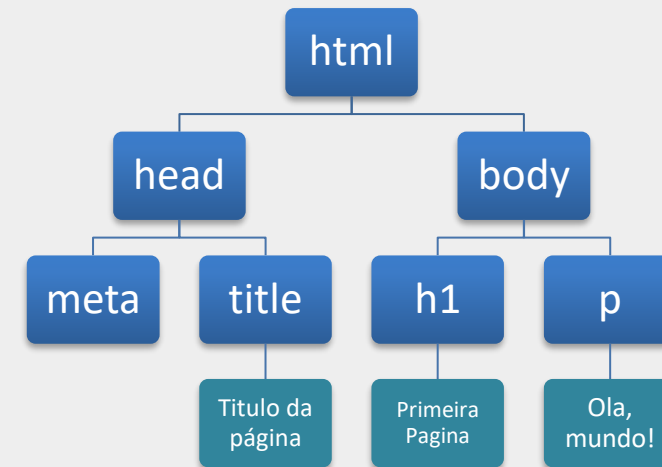
  <head>
    <meta charset="UTF-8">
    <title>Titulo da Pagina</title>
  </head>

  <body>

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

  </body>

</html>
```



Árvore DOM (simplificada)

Manipulação da Árvore DOM

- Adicionar/modificar **conteúdo** aos elementos
- 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 na verdade 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>

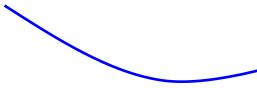
  </head>

  <body>

    ...

  </body>

</html>
```



Arquivo JS

```
/* arquivoJavaScript.js */
```

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)
```

Declarando Variáveis

var

- 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

- 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

- 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>
```

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 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.

document (ou `window.document`)

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

Métodos para E/S

<code>window.alert</code>	exibe uma caixa de diálogo para mensagens (botão Ok)
<code>window.confirm</code>	exibe uma caixa de diálogo para confirmação (Ok/Cancelar)
<code>window.prompt</code>	exibe uma caixa de diálogo para entrada de texto
<code>document.write</code>	adiciona conteúdo no documento HTML
<code>console.log</code>	registra conteúdo de log na janela 'console' do navegador
<code>console.warn</code>	registra msg de <i>warning</i> na janela 'console' do navegador
<code>console.error</code>	registra msg de <i>erro</i> na janela '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

Tratando 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 qdo 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

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'

```
document.addEventListener("DOMContentLoaded", funcaoIniciaPagina);
```

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 já
definida e indicada
para tratar evento*

Arrow Function =>

- Define funções sem utilizar a palavra **function**
- Sintaxe mais curta utilizando '**=>**'
- 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 =>

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);
```

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 *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
 - `x = 5 + 5;` `// x terá o valor 10`
 - `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

```
1 == true;    // true;  
1 === true;   // false;  
10 == "10"    // true;  
10 === "10"   // false;
```

Vetores (*arrays*)

- Elementos colocados entre colchetes, separados por vírgula

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

- Primeiro elemento tem índice 0

```
let primeiroPar = pares[0];
```

- Tratado como objeto

```
let nroElementos = pares.length;
```

- Elementos de diferentes tipos

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

- Pode ser iniciado com vazio

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

Percorrendo Vetor 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 Vetor com Método *forEach*

```
let pares = [2, 4, 6, 8];  
let soma = 0;  
pares.forEach( function (elemento) {  
    soma += elemento;  
});
```

Percorrendo vetor com método `forEach` e função anônima

Percorrendo Vetor com Método *forEach*

```
let pares = [2, 4, 6, 8];  
let soma = 0;  
pares.forEach( elemento => {  
    soma += elemento;  
});
```

Percorrendo vetor com método *forEach* e *arrow function*

Percorrendo Vetor com Método *forEach*

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

Percorrendo vetor: *forEach*, *arrow function* e única operação

Vetores – Métodos e Propriedades

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

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

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 outras propriedades e métodos

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

Template Literals (ou Template String)

- Strings definidas com o caracter 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
- Exemplo:

```
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
- Comumente definido utilizando chaves { }
- Lista de pares do tipo *propriedade* : *valor*
- Criado como instância da classe **Object**

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

```
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

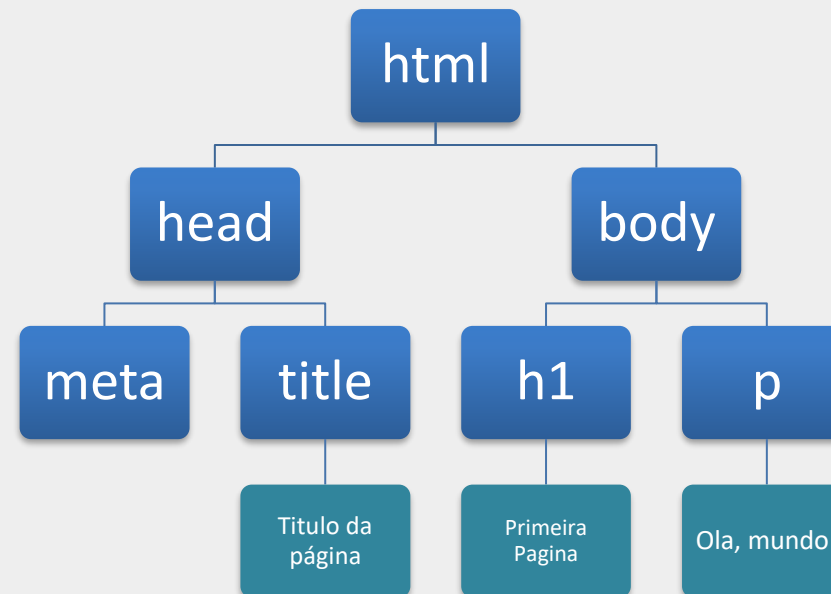
Document Object Model - DOM

```
<!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 a página, o navegador faz uma varredura do HTML e monta uma estrutura de dados internamente denominada árvore DOM, que é uma representação em memória de toda a estrutura do documento HTML. Essa estrutura é utilizada para manipular o documento HTML dinamicamente, utilizando programação, com a DOM API.

Tipos de Nós na Estrutura DOM

- Nó tipo *Element* - nó (objeto) representando um elemento HTML
- Nó tipo *Text* - nó representando um conteúdo textual
- Nó tipo *Comment* - nó representando um comentário

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

Manipulação da Árvore DOM

`document.querySelector`

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

Manipulação da Árvore DOM

`document.querySelector`

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

Retorna o nó correspondente ao primeiro elemento h1 na página

Manipulação da Árvore DOM

`document.querySelector`

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

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

Manipulação da Árvore DOM

`document.querySelector`

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

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

Manipulação da Árvore DOM

`document.querySelectorAll`

- 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

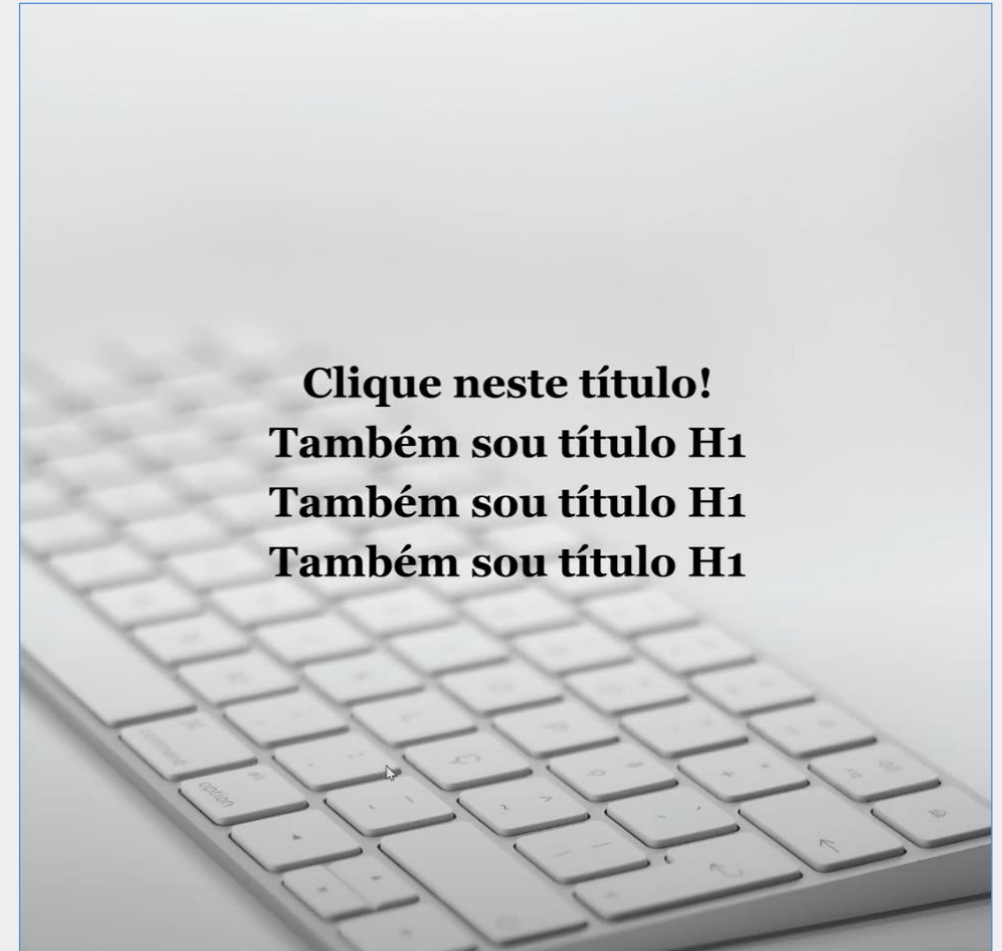
Manipulação da Árvore DOM

`document.querySelectorAll`

```
// 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>
```



Manipulação da Árvore DOM

`document.getElementById`

- busca um único elemento utilizando o seu id

`document.getElementsByName`

- busca os elementos pelo valor do atributo name

`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");
```

Acesso ao Conteúdo dos Elementos

- Propriedade **textContent**
- Propriedade **innerText**
- Propriedade **innerHTML**

Acesso ao Conteúdo dos Elementos

Propriedade **textContent**

- Se o conteúdo 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 por um texto

Propriedade **innerText**

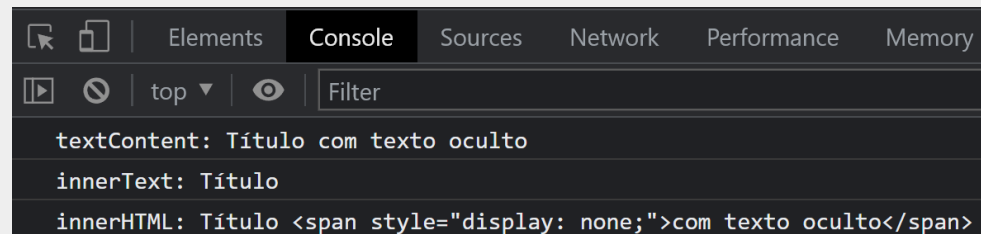
- Semelhante a **textContent**, porém leva em consideração estilos CSS
- Não retorna conteúdo relativo a elementos **que não aparecem na página**

Propriedade **innerHTML**

- Retorna o conteúdo do elemento e seus descendentes após serialização 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>
```



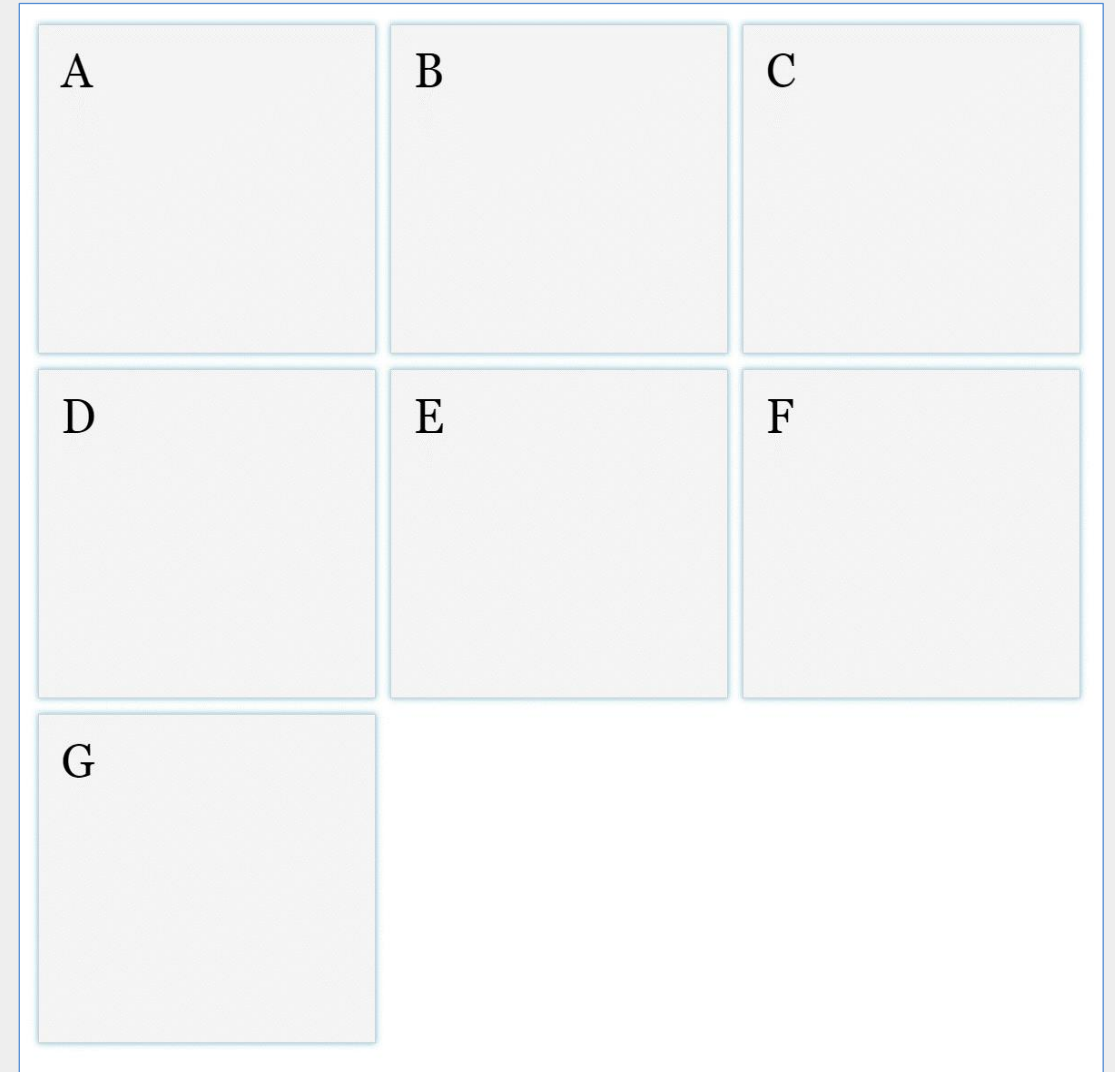
Alterando Estilos CSS de forma *inline*

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

CSS	JavaScript
color	node.style.color
font-family	node.style.fontFamily
background-color	node.style.backgroundColor

Alterando Estilos CSS de forma *inline*

```
22 <body>
23
24 <main>
25   <article>A</article>
26   <article>B</article>
27   <article>C</article>
28   <article>D</article>
29   <article>E</article>
30   <article>F</article>
31   <article>G</article>
32 </main>
33
34 <script>
35
36   window.onload = function () {
37     const artigos = document.querySelectorAll("article");
38     for (let artigo of artigos) {
39       artigo.onclick = () => artigo.style.visibility = 'hidden';
40     }
41   }
42
43 </script>
44
45 </body>
```



Manipulando Atributos

A maioria dos atributos pode ser acessada indiretamente utilizando propriedades de mesmo nome dos respectivos objetos 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
- Em alguns casos 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.firstChildElementChild`
 - 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 de nós filhos
- inclui nós de texto, nós de comentário, etc.
- `node.children` - retorna coleção dos nós filhos do **tipo elemento**
- `node.appendChild` - acrescenta um nó filho no final da lista de filhos
- `node.removeChild` - remove um nó filho da lista de filhos

Manipulação da Árvore DOM

- `node.remove` - remove o nó da lista de filhos do nó pai
- `node.parentNode` - retorna o nó pai
- `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**
- `node.cloneNode` - clona o objeto correspondente ao nó
- `document.createElement` - cria um novo nó do tipo **element**

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     window.onload = function () {
33       const botaoAdicionar = document.querySelector("button");
34       botaoAdicionar.addEventListener("click", adicionaInteresse);
35     }
36
37     function adicionaInteresse() {
38       const campoInteresse = document.querySelector("#interesse");
39       const listaInteresses = document.querySelector("ol");
40
41       const novoLi = document.createElement("li");
42       novoLi.textContent = campoInteresse.value;
43       listaInteresses.appendChild(novoLi);
44       campoInteresse.value = '';
45     }
46   </script>
47
48 </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 X
2. Esportes X
3. Viagens X
4. Filmes X

Collections do Objeto document

Propriedades especiais que dão acesso direto a alguns elementos

- `document.forms` - retorna coleção de todos os formulários (<form>)
- `document.images` - retorna coleção de todas as imagens ()
- `document.anchors` - retorna coleção de todos os links (<a>)

Exemplos de Uso de *document.forms*

```
<form name="cadastro">  
  Produto: <input name="produto">  
</form>
```

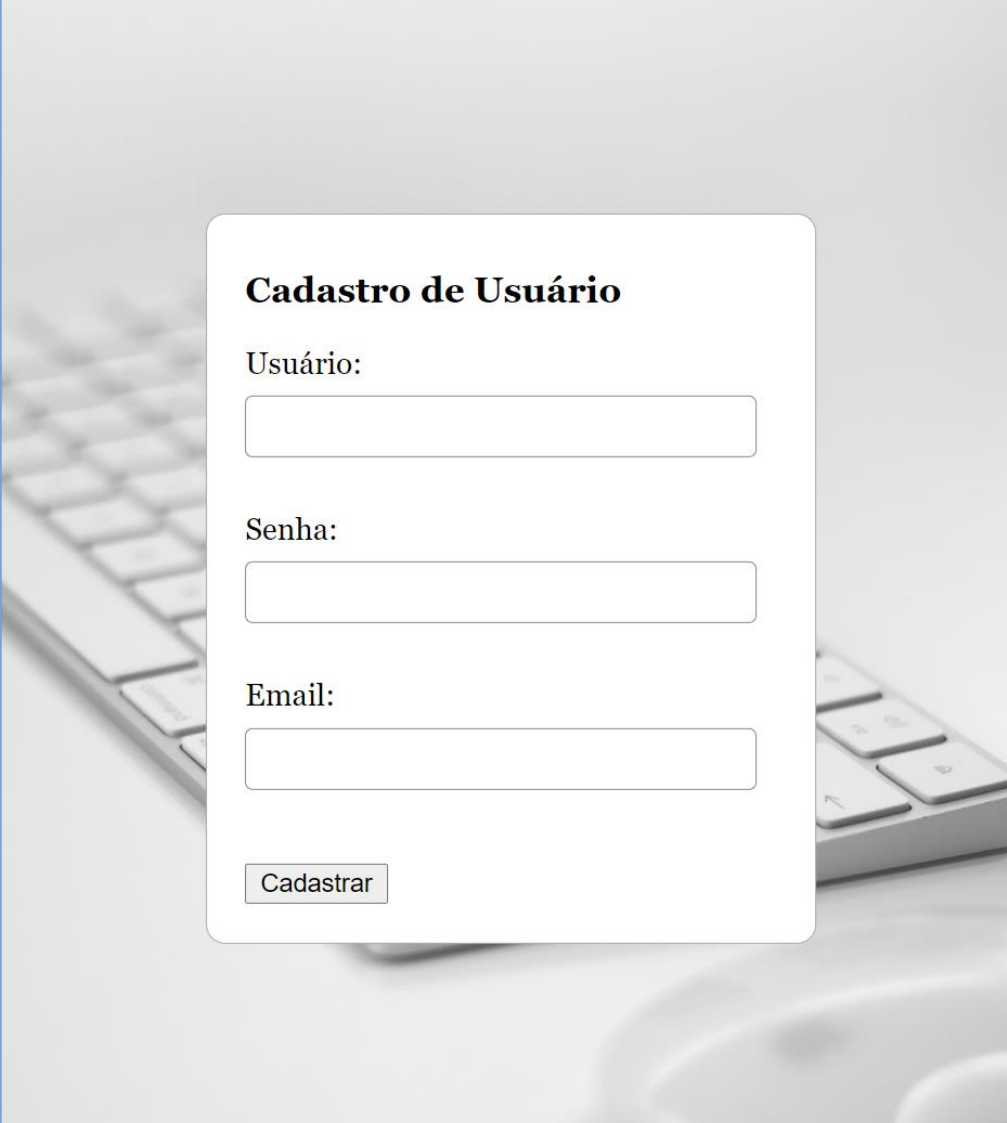
```
const campoProduto = document.forms.cadastro.produto;  
const valorDoCampo = campoProduto.value;
```

Outras Formas

- `const campoProduto = document.forms["cadastro"]["produto"];`
- `const campoProduto = document.forms["cadastro"].elements.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 user registration form. It features a white rounded rectangle centered on a blurred background of a laptop keyboard. The form has a title 'Cadastro de Usuário' in bold. Below the title are three input fields: 'Usuário:', 'Senha:', and 'Email:'. Each label is followed by its respective input field. At the bottom of the form is a button labeled 'Cadastrar'.

Validação de Formulário - Exemplo Simples

```
10 window.onload = function () {
11     document.forms.formCadastro.onsubmit = validaForm;
12 }
13
14 function validaForm (e) {
15     let form = e.target;
16     let formValido = true;
17
18     const spanUsuario = form.usuario.nextElementSibling;
19     const spanSenha = form.senha.nextElementSibling;
20     const spanEmail = form.email.nextElementSibling;
21
22     spanUsuario.textContent = "";
23     spanSenha.textContent = "";
24     spanEmail.textContent = "";
25
26     if (form.usuario.value === "") {
27         spanUsuario.textContent = 'O usuário deve ser preenchido';
28         formValido = false;
29     }
30
31     if (form.senha.value === "") {
32         spanSenha.textContent = 'A senha deve ser preenchida';
33         formValido = false;
34     }
35
36     if (form.email.value === "") {
37         spanEmail.textContent = 'O email deve ser preenchido';
38         formValido = false;
39     }
40
41     return formValido;
42 }
```

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/>
- **JavaScript and JQuery: Interactive Front-End Web Development**, Jon Duckett.