Programação para Internet

Módulo 5

Páginas Interativas com JavaScript – Gestão da Informação

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

Conteúdo da Módulo

- Introdução
- Recursos Básicos da Linguagem
- Document Object Model – Árvore DOM
- Tratamento de Eventos
- Manipulação da Árvore DOM

O que é JavaScript?

- Linguagem de script de alto nível integrada nos navegadores
- Permite prover interatividade e dinamismo a websites
- Permite programar o comportamento da página Web na ocorrência de eventos
- Permite alterar o documento HTML dinamicamente
- 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
- JavaScript é as vezes referenciada pela abreviação JS
- Não confundir com a linguagem de programação Java

O que posso fazer com JavaScript?

- Modificar o conteúdo dos elementos HTML da página
- Adicionar novos elementos HTML na página dinamicamente
- Remover elementos HTML da página dinamicamente
- Modificar os atributos dos elementos dinamicamente
- Modificar os estilos CSS dos elementos dinamicamente
- Fazer requisições HTTP assíncronas
- Validar formulários etc.

Inserindo JavaScript de Forma Embutida no HTML

```
<html>

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

    <body>
        ...
    </body>

</html>
```

```
<html>

    <head>
        ...
    </head>

    <body>
        ...

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

</html>
```

O código JavaScript pode ser inserido de forma embutida dentro do próprio arquivo HTML utilizando o elemento <script> da HTML. O código pode ser inserido na região de cabeçalho, no corpo da página ou até mesmo depois da tag </body>.

JavaScript em Arquivo Separado

Arquivo HTML

```html
<html>

  <head>

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

  </head>

  <body>

    ...

  </body>

</html>
```

Arquivo JavaScript (.js)

```javascript
/* arquivoJavaScript.js */

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

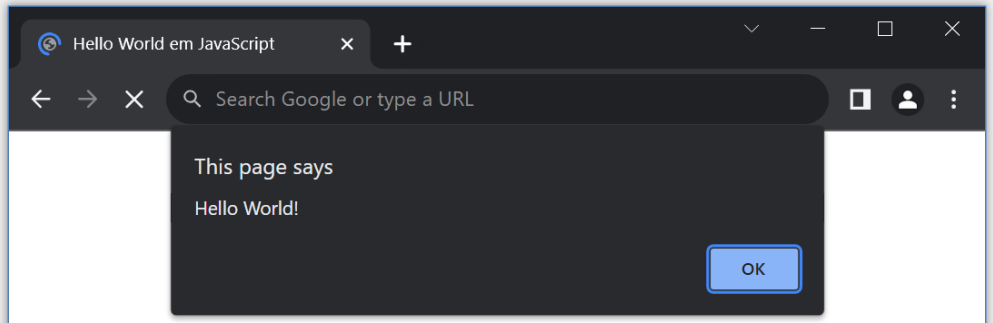
Uma forma melhor de inserir o código JavaScript é utilizar um arquivo externo e referenciá-lo na tag <script> com o atributo src. Também há a possibilidade de inserir JavaScript como **módulo**, mas esse assunto não será abordado neste material (módulos são inseridos com <script type="module">)

Vantagens do JavaScript em Arquivo Separado

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

Hello World em JavaScript

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

<head>
  <meta charset="UTF-8">
  <title>Hello World em JavaScript</title>
</head>

<body>

  <h1>Minha primeira página com JavaScript</h1>
  <p>Lorem ipsum dolor sit amet.</p>

  <script>
    alert("Hello World!");
  </script>
</body>

</html>
```



Neste exemplo, a janela de alerta com a mensagem Hello World! é apresentada ao usuário durante o carregamento da página. O conteúdo propriamente dito (título e parágrafo) será mostrado depois que o usuário clicar no botão OK e a página terminar de ser carregada.

Exemplo de Código JavaScript para Alterar os Estilos Dinamicamente

```
<head>
  <style>
    .destaca {
      box-shadow: 0 0 20px red;
    }
  </style>
</head>
<body>
  <h1>Passe o cursor sobre a imagem</h1>
  <img src="images/logoUFU.png">
  <script>
    const img = document.querySelector("img");
    img.onmouseenter = () => img.classList.add("destaca");
    img.onmouseleave = () => img.classList.remove("destaca");
  </script>
</body>
```

Recursos Básicos 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
}
```

```
for (let item of array)
{
   // 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 onde é declarada
- Pode ser acessada e atualizada apenas dentro do bloco
- Não pode ser redeclarada no mesmo bloco

const nomeDaConstante = valor

- Semelhante a let, porém a variável não pode ser atualizada
- Deve ser inicializada no momento da declaração

Exemplos de Declarações 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 de Atribuição

Operador	Significado
+	Adição (e concatenação)
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão inteira
++	Incremento
--	Decremento
=	Atribuição
+=	Atribuição com soma ou concatenação
-=	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
- Permite 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’
let z = true + '5';  // z terá a string ‘true5’
```

Objetos window, navigator, document e console

window

- Objeto global que representa a aba do navegador contendo 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 página for carregada, fechada etc.
  - Mostrar mensagens de alerta: window.alert("mensagem");

navigator (ou window.navigator)

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

document (ou window.document)

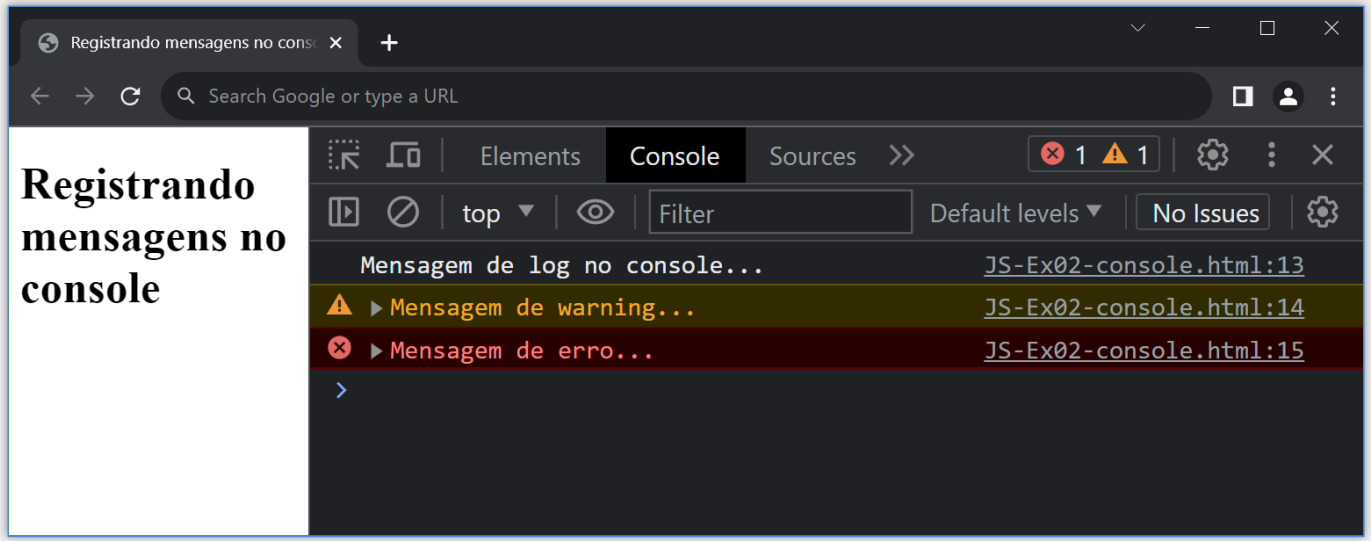
- Representa o documento HTML carregado na aba do navegador
- Possibilita a manipulação da árvore DOM

console

- Dá acesso à janela de console de depuração do navegador

Registrando Mensagens na Janela de Console do Navegador

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

<head>
  <meta charset="UTF-8">
  <title>Registrando mensagens no console</title>
</head>

<body>
  <h1>Registrando mensagens no console</h1>

  <script>
    console.log('Mensagem de log no console...');
    console.warn('Mensagem de warning...');
    console.error('Mensagem de erro...');
  </script>
</body>

</html>
```



Strings

- Podem ser definidas com aspas simples ou duplas

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

- Caracteres podem ser acessados por colchetes ou pelo método chatAt

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

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

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

- É possível fazer o escape de caracteres especiais com a contrabarra

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

- Strings são objetos com propriedades e métodos

```
msg.length, msg.indexOf('alright'), msg.substr(0,2), msg.split etc.
```

- São imutáveis (não podem ser alteradas)

Template Literals (ou Template Strings)

- São strings definidas com o caractere crase: `minha string`
- Suporta fácil interpolação de variáveis e expressões com ${ }
- 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}`);
```

Arrays

- Os elementos são colocados entre colchetes, separados por vírgula

```
let pares = [2, 4, 6, 8];
let primeiroPar = pares[0]; // 1º elemento
let nroElementos = pares.length; // tamanho do array
```

- É possível ter elementos de tipos diferentes

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

- O array pode ser iniciado com vazio

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

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);
};
```

OBS: Há também a estrutura for..in que permite iterar sobre as propriedades enumeráveis de um objeto. Porém, for..in não poderia ser utilizada no contexto acima. Com for..in cada item no laço acima receberia o **índice** do item no *array*, mas não o **elemento** propriamente dito.

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
```

A propriedade "motor-hp", por ter um caractere especial, **não poderia** ser acessada utilizando a notação com o ponto (carro.motor-hp). Para esses casos deve-se utilizar a notação com colchetes.

Declaração de Funções

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

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

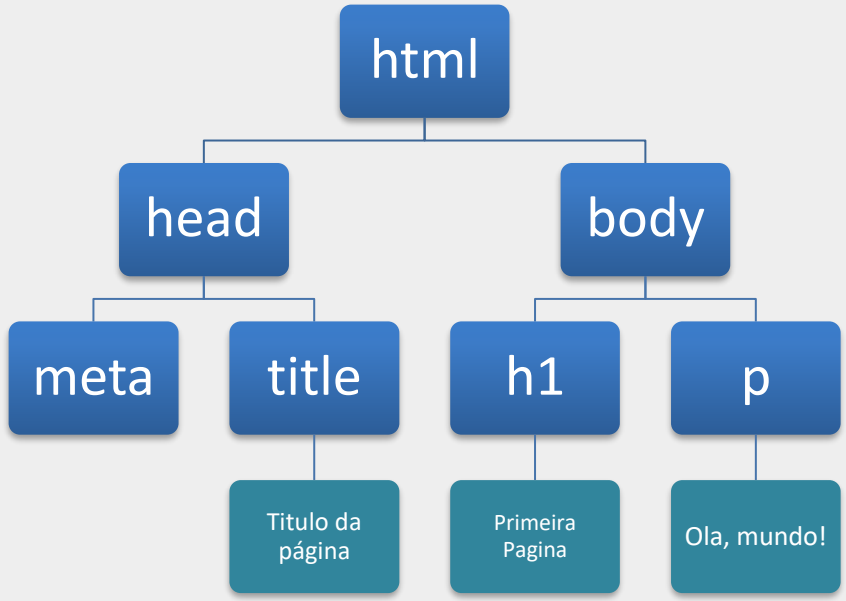
```
let maior = max(2, 5);
```

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

Manipulação da 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 árvore DOM permite a manipulação do documento HTML dinamicamente, utilizando programação, com a **DOM API** e a JavaScript.

Busca na Árvore DOM

```
document.querySelector("seletor CSS")
```

- Faz uma busca na árvore DOM utilizando a string de seleção CSS e retorna o primeiro nó na árvore do tipo Element que atende à seleção
- Retorna `null` caso não haja correspondências
- Nenhum nó é retornado caso o seletor inclua pseudo-elementos

OBS: o método querySelector é definido no objeto document e também nos nós do tipo Element. Portanto, é possível fazer uma busca em toda a árvore DOM ou apenas em um ramo da árvore que inicia a partir de um nó de elemento.

document.querySelector – Exemplo

```
<body>
  <h1>Primeiro Título</h1>
  <h1>Segundo Título</h1>
  <script>
    const firstNodeH1 = document.querySelector("h1");
    alert(firstNodeH1.textContent); // mostra 'Primeiro Título'
    firstNodeH1.textContent = "Título alterado com JavaScript";
  </script>
</body>
```

A propriedade textContent do nó permite acessar e alterar o conteúdo do respectivo elemento HTML

document.querySelector – Exemplos Adicionais

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");
```

Retorna o nó correspondente ao primeiro 'input' com name="sexo" selecionado

```
const nodeRadio = document.querySelector('input[name="sexo"]:checked');
```

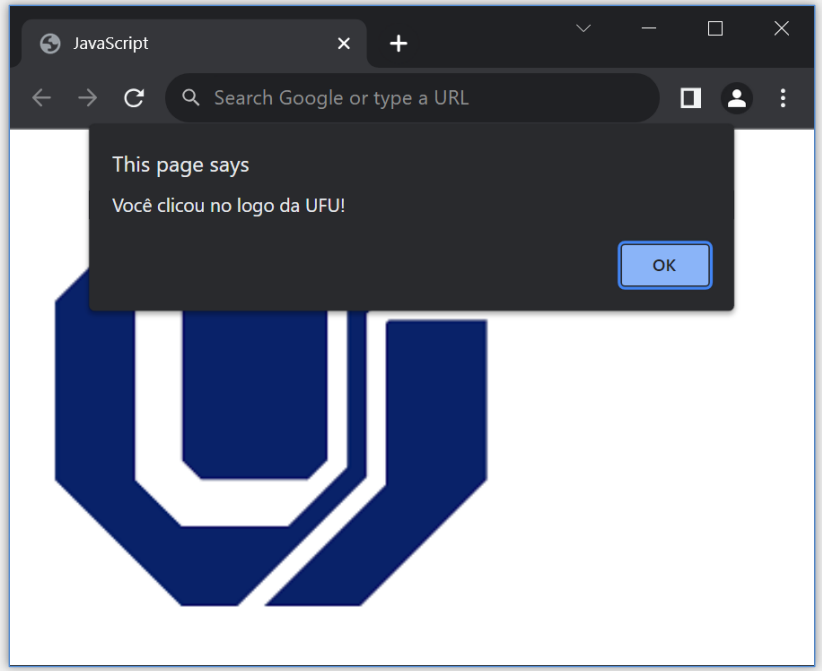
Tratamento de Eventos

- JavaScript é baseada em eventos
- É possível executar funções na ocorrência de eventos como “clique em botão”, “seleção de item”, “rolagem da página” etc.
- As funções para tratar os eventos pode ser registradas de duas formas:
  - Utilizando propriedades de eventos dos nós
  - Ou utilizando o método addEventListener

Tratando Eventos com Propriedades de Eventos

- Propriedades de eventos são propriedades dos objetos que permitem a indicação de uma função para tratar os eventos nos objetos
- Essas propriedades começam com “on” seguido do nome do evento, ou seja, onclick, onmouseover, onkeyup, onsubmit etc.
- Basta atribuir o nome da função à propriedade (sem parênteses). A função será chamada automaticamente quando ocorrer o evento no objeto

```
<body>
  <img src="images/logoUFU.png" id="imgUFU">
  <script>
    // Função para tratar o evento clique na 1ª imagem
    function trataClique() {
      alert("Você clicou no logo da UFU!");
    }

    // Registra a função trataClique para tratar o evento 'click' na imagem.
    // Registro feito com a propriedade de evento onclick do nó.
    const nodeImage = document.querySelector("#imgUFU");
    nodeImage.onclick = trataClique;
  </script>
</body>
```



Tratando Eventos com addEventListener

- Outra forma de registrar funções para tratar eventos é por meio do método addEventListener do objeto
- Esse método tem dois parâmetros principais:
  - O primeiro é o nome do evento (não tem o "on")
  - O segundo é a função para tratar o evento
  - Ex.: object.addEventListenerr("click", trataClique);

```
<body>
  <img src="images/logoUFU.png" id="imgUFU">
  <script>
    // Função para tratar o evento clique na imagem
    function trataClique() {
      alert("Você clicou no logo da UFU!");
    }

    // Registra a função trataClique para tratar o evento 'click' na imagem.
    // Registro feito com o método addEventListener do nó.
    const nodeImage = document.querySelector("#imgUFU");
    nodeImage.addEventListener('click', trataClique);
  </script>
</body>
```

Código análogo ao do slide anterior, porém utilizando o método addEventListerner, ao invés da propriedade de evento do objeto.

OBS: repare que não há parênteses () depois de trataClique na linha do addEventListener, pois **não estamos chamando** a função, mas apenas **indicando-a** para ser chamada na ocorrência do evento.

addEventListener tem ainda um 3º parâmetro opcional não abordado neste material.

Tratando Eventos com addEventListener

- Com addEventListener é possível registrar múltiplas funções para tratar o evento no objeto
- Quando o evento ocorrer, as funções serão chamadas na ordem em que foram registradas

```
<body>
  <img src="images/logoUFU.png" id="imgUFU">
  <script>
    function funcao1() {
      alert("Você clicou no logo da UFU!");
    }
    function funcao2() {
      alert("Obrigado!");
    }
    const nodeImage = document.querySelector("#imgUFU");
    nodeImage.addEventListener('click', funcao1);
    nodeImage.addEventListener('click', funcao2);
  </script>
</body>
```

Quando o usuário clicar na imagem aparecerá primeiramente a mensagem "Você clicou...". Posteriormente, aparecerá "Obrigado".

Funções Anônimas

- Funções tratadoras de eventos podem ser definidas no momento em que são indicadas para tratar o evento
- Isto é possível utilizando funções anônimas, como apresentado no exemplo a seguir

```
<body>
  <img src="images/logoUFU.png" id="imgUFU">
  <script>
    // Função anônima indicada para tratar evento
    // ao mesmo tempo em que é definida
    const nodeImage = document.querySelector("#imgUFU");
    nodeImage.addEventListener('click', function () {
      alert("Você clicou no logo da UFU!");
    });
  </script>
</body>
```

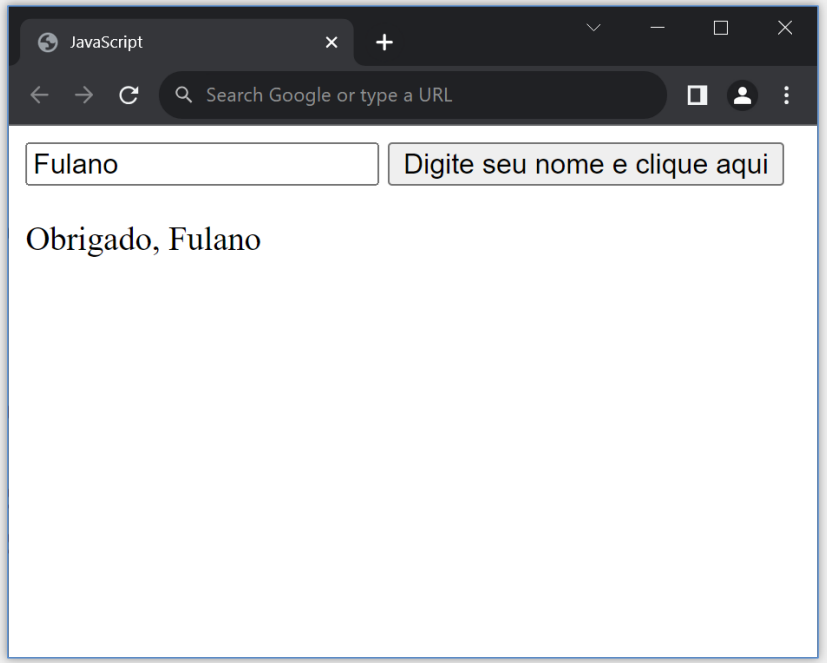
Funções Anônimas

```
<body>
  <img src="images/logoUFU.png" id="imgUFU">
  <script>
    // Função anônima indicada para tratar evento
    // ao mesmo tempo em que é definida
    const nodeImage = document.querySelector("#imgUFU");
    nodeImage.onclick = function () {
      alert("Você clicou no logo da UFU!");
    };
  </script>
</body>
```

Função anônima sendo utilizada em conjunto com propriedade de evento

Tratando Eventos – Exemplo Adicional

```
<body>
  <input type="text" name="nome">
  <button type="button">Digite seu nome e clique aqui</button>
  <p></p>

  <script>
    const botao = document.querySelector("button");
    botao.onclick = function () {
      const campoNome = document.querySelector("input");
      const parSaida =  document.querySelector("p");
      parSaida.textContent = 'Obrigado, ' + campoNome.value;
    };
  </script>
</body>
```

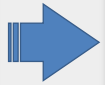


A propriedade value do nó representando o elemento HTML <input> permite acessar o valor do campo (texto preenchido pelo usuário).

Arrow Function =>

- Permite definir funções sem utilizar a palavra function
- A definição é feita de forma abreviada utilizando os caracteres '=>'

```
objeto.onclick = function () {
      // operações
}
```



```
objeto.onclick = () => {
      // operações
}
```

Função anônima definida com a palavra function

Declaração correspondente utilizando arrow function

OBS: arrow function não substitui a definição tradicional de funções em todas as situações.

Arrow Function – Exemplos Adicionais

- Função com uma única declaração dispensa as chaves

```
objeto.onclick = () => alert('Você clicou em...');
```

- Arrow function também pode ter parâmetros

```
objeto.onclick = (e) => alert('Objeto clicado: ' + e.target);
```

- Arrow function com um único parâmetro não precisa dos parênteses

```
objeto.onclick = e => alert('Objeto clicado: ' + e.target);
```

O parâmetro `e` é um objeto contendo informações do evento disparado. Por exemplo, `e.target` contém uma referência para o objeto que disparou o evento (no caso acima, o objeto em particular (botão, imagem etc.) que recebeu o click do usuário).

Outros Eventos Comuns

Evento	Propriedade	Quando Ocorre o Evento
mouseenter	onmouseenter	Quando o cursor do mouse entra na região do elemento
mouseleave	onmouseleave	Quando o cursor sai da região do elemento
keydown/keyup	onkeydown/onkeyup	Quando o usuário pressiona/libera alguma tecla
focus	onfocus	Quando o elemento recebe o foco (clique em campo do formulário)
blur	onblur	Quando o elemento perde o foco (clique fora do campo)
change	onchange	Quando um campo de formulário tem o valor alterado (em campos textuais é disparado apenas na perda do foco)
submit	onsubmit	Quando o formulário é submetido
load	onload	Quando a página termina de ser carregada por completo
DOMContentLoaded	--	Quando a árvore DOM termina de ser carregada

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

Onde está o erro?

Arquivo HTML

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Página com erro</title>
  <script src="meuscript.js"></script>
</head>
<body>
  <h1>Hello!</h1>
  <button>Clique aqui!</button>
</body>
</html>
```

Arquivo meuscript.js

```
1
2  const botao = document.querySelector("button");
3  botao.onclick = () => alert("Obrigado!");
4
```

Onde está o erro?

Arquivo HTML

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Página com erro</title>
  <script src="meuscript.js"></script>
</head>
<body>
  <h1>Hello!</h1>
  <button>Clique aqui!</button>
</body>
</html>
```

Arquivo meuscript.js

```
1
2  const botao = document.querySelector("button");
3  botao.onclick = () => alert("Obrigado!");
4
```

Quando o navegador executar o código JavaScript, a árvore DOM estará incompleta e não possuirá o nó correspondente ao elemento button. Dessa forma, o método querySelector retornará null. Isso acontece porque o arquivo JavaScript está sendo referenciado no início do arquivo HTML, dentro da região de cabeçalho, e será processado antes do HTML restante ser carregado.

Solução utilizando DOMContentLoaded

Arquivo HTML

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Página com erro</title>
  <script src="meuscript.js"></script>
</head>
<body>
  <h1>Hello!</h1>
  <button>Clique aqui!</button>
</body>
</html>
```

Arquivo meuscript.js

```js
document.addEventListener('DOMContentLoaded', () => {
  const botao = document.querySelector("button");
  botao.onclick = () => alert("Obrigado!");
});
```

Uma solução é utilizar o evento DOMContentLoaded para garantir que o código seja executado apenas depois que a árvore DOM for completamente carregada.

Outra possibilidade é utilizar o atributo defer na tag <script>, para dizer ao navegador que o script deve ser executado depois que o documento HTML terminar de ser carregado (<script src="meuscript.js" defer></script>)

Busca na Árvore DOM com querySelectorAll

`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

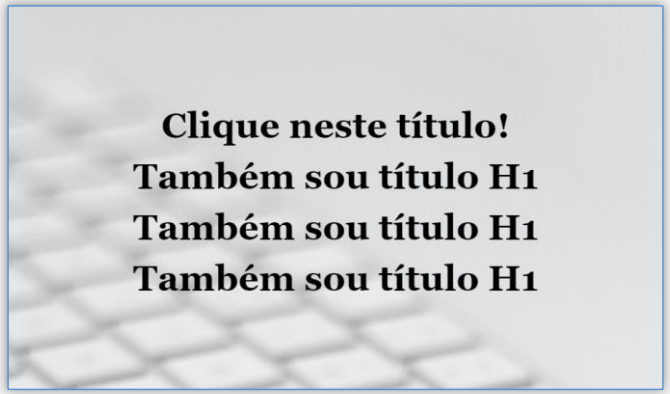
Busca na Árvore DOM com querySelectorAll

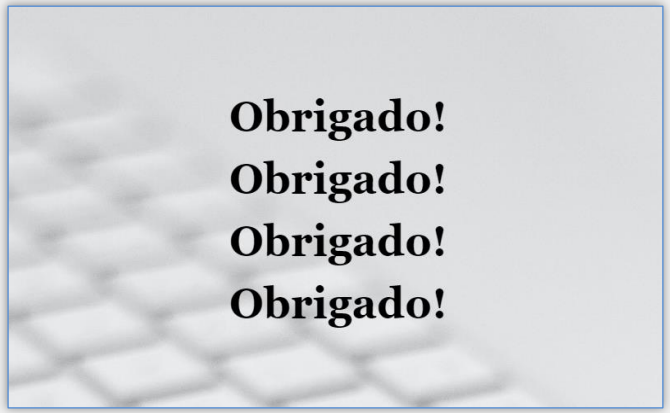
```
...
<body>
  <h1>Título 1</h1>
  <h1>Título 2</h1>
  <h1>Título 3</h1>
</body>
...
```

```
// retorna os nós correspondentes a todos os elementos h1 da página
const nodesH1 = document.querySelectorAll("h1");

// mostra "Título 1", "Título 2" e "Título 3"
for (let node of nodesH1) {
    console.log(node.textContent);
}
```

Manipulação da Árvore DOM – Exemplo

```
<main>
  <h1>Clique neste título!</h1>
  <h1>Também sou título H1</h1>
  <h1>Também sou título H1</h1>
  <h1>Também sou título H1</h1>
</main>
<script>
  const nodeH1 = document.querySelector("h1");
  nodeH1.addEventListener("click", function () {
    const nodesH1 = document.querySelectorAll("h1");
    for (let node of nodesH1)
      node.textContent = "Obrigado!";
  });
</script>
```





Neste exemplo, quando o usuário clicar no **primeiro** título <h1>, **todos** os títulos <h1> terão seu conteúdo alterado para "Obrigado!"

Manipulação da Árvore DOM – Exemplo

```
<main>
  <h1>Clique neste título!</h1>
  <h1>Clique neste título!</h1>
  <h1>Clique neste título!</h1>
  <h1>Clique neste título!</h1>
</main>
<script>
  const nodesH1 = document.querySelectorAll("h1");
  for (let node of nodesH1)
    node.onclick = () => node.textContent = "Obrigado!";
</script>
```

Neste exemplo, quando o usuário clicar em **qualquer** título <h1>, seu **respectivo** texto será alterado para "Obrigado!". Observe que uma função tratadora de evento é registrada para cada nó, que fará a modificação do texto do próprio nó.

Outras Formas de Realizar Buscas na Árvore DOM

document.getElementById

- busca um nó de elemento utilizando o seu id

document.getElementsByName

- busca nós de elementos pelo valor do atributo name do elemento

document.getElementsByTagName

- busca nós de elementos pelo nome da tag HTML, como img, h1, etc.

document.getElementsByClassName

- busca nós de elementos pelo valor do atributo class do elemento

Conteúdo Complementar

Acesso ao Conteúdo dos Elementos

Propriedade textContent

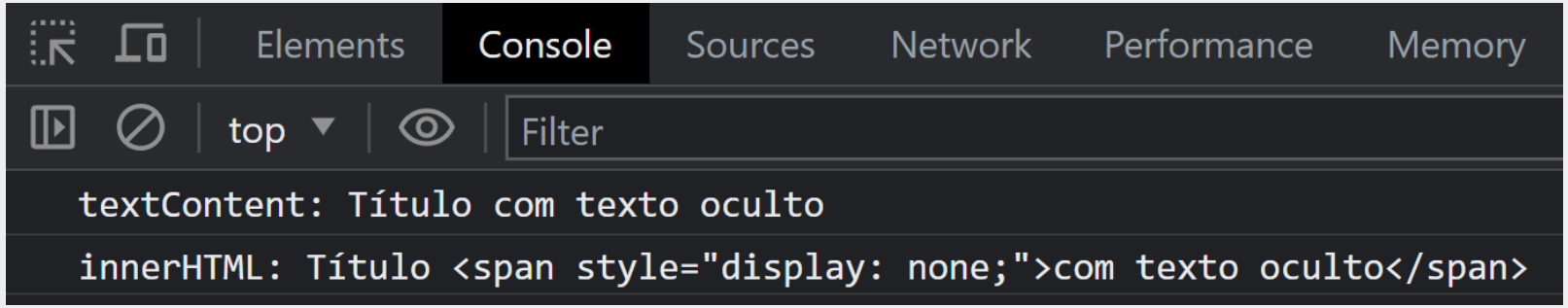
- Se o conteúdo do elemento é textual, retorna esse texto
- Se o elemento possui filhos, retorna a **concatenação** do textContent dos filhos
- Uma alteração do valor removerá todos os nós filhos e **substituirá** pelo novo texto

Propriedade 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 novos nós descendentes na estrutura DOM
- **OBS**: possibilidade de ataques XSS e desempenho inferior ao de textContent.

Acessando o Conteúdo com textContent e innerHTML

```
<main>
  <h1>Título <span style="display: none;">com texto oculto</span></h1>
</main>
<script>
  const nodeH1 = document.querySelector("h1");
  console.log("textContent: " + nodeH1.textContent);
  console.log("innerHTML: " + nodeH1.innerHTML);
</script>
```



Modificando o Conteúdo com textContent e innerHTML

```
<main>
  <h2></h2>
  <h2></h2>
</main>
<script>
  const nodes = document.querySelectorAll("h2");
  const conteudo = 'Titulo com <span style="color: red;">texto vermelho</span>';
  nodes[0].textContent = conteudo;
  nodes[1].innerHTML = conteudo;
</script>
```

Aspectos de Segurança – Propriedade innerHTML e Ataques XSS

```html
<main>
  <input type="text">
  <button>Digite algo e clique aqui</button>
  <p></p>
</main>
<script>
  const botao = document.querySelector("button");
  botao.onclick = function () {
    const valorUsuario = document.querySelector("input").value;
    const pSaida = document.querySelector("p");
    pSaida.innerHTML = valorUsuario;  // vulnerável a XSS!!
  }
</script>
```

Este é um exemplo de página vulnerável a ataques XSS devido ao uso inadequado da propriedade innerHTML, permitindo que um usuário malicioso execute o seu próprio código JavaScript na página. Por exemplo, se o usuário digitar no campo textual a string <img src="inexistente.jpg" onerror="alert('AHA')">, o código JavaScript inserido na propriedade onerror será executado, pois o navegador não encontrará a imagem "inexistente.jpg".

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

Porém alguns atributos são acessados de forma diferenciada

Atributo HTML	JavaScript
`for`	`node.htmlFor`
`class`	`node.className`
`data-matricula`	`node.dataset.matricula`
`data-num-matricula`	`node.dataset.numMatricula`
	`node.dataset["numMatricula"]`

Manipulando Atributos

```
<body>
  <main>
    <img src="images/logoUFU.png">
  </main>
  <script>
    const nodeImage = document.querySelector("img");
    nodeImage.onclick = function () {
      nodeImage.src = "images/logoFacom.png";
      nodeImage.alt = "Faculdade de Computação"
    }
  </script>
</body>
```

Quando o usuário clicar sobre a imagem ela será trocada por outra.

Manipulando Atributos para Alterar Estilos CSS

- Uma forma de alterar os estilos CSS de um elemento é por meio do atributo `style` do respectivo nó na árvore DOM
- Os nomes das propriedades segue padrão **CamelCase**

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

Ao utilizar a propriedade style do nó, a alteração do CSS ocorre de forma "inline".

Manipulando Atributos para Alterar Estilos CSS

```
<main>
  <h1>Título</h1>
  <h1>Título</h1>
  <h1>Título</h1>
</main>
<script>
  const nodesH1 = document.querySelectorAll("h1");
  for (let node of nodesH1) {
    node.onclick = () => node.style.visibility = 'hidden';
  }
</script>
```

Neste exemplo, o clique em um título fará com que ele fique oculto.
OBS: algumas propriedades CSS cujos valores são calculados dinamicamente pelo navegador (ex. margin, padding, width etc.) não podem ter o valor corrente resgatado utilizando node.style.nomePropriedade. Nesses casos pode ser necessário utilizar a função getComputedStyle.

Manipulando Atributos para Alterar Estilos CSS

- Uma forma melhor de alterar os estilos dos elementos dinamicamente é adicionando ou removendo classes CSS
- Pode-se utilizar:
  - node.classList.add para adicionar uma classe CSS ao elemento, ou
  - node.classList.remove para remover uma classe CSS do elemento

Manipulando Atributos para Alterar Estilos CSS

```
<head>
  <style>
    .destaca {
      box-shadow: 0 0 20px red;
    }
  </style>
</head>
<body>
  <h1>Passe o cursor sobre a imagem</h1>
  <img src="images/logoUFU.png">
  <script>
    const img = document.querySelector("img");
    img.onmouseenter = () => img.classList.add("destaca");
    img.onmouseleave = () => img.classList.remove("destaca");
  </script>
</body>
```

Criando e Adicionando Novos Nós na Árvore DOM

`document.createElement("nomeDoElementoASerCriado")`

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

`node.appendChild(novoNo)`

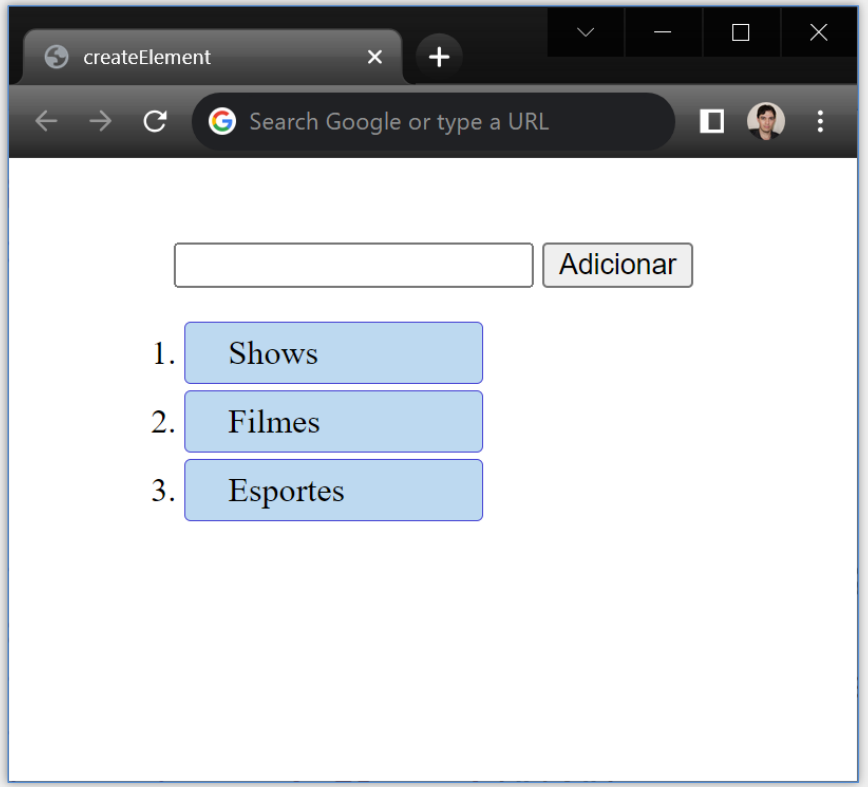
- acrescenta um nó filho no final da lista de filhos

`node.removeChild(noFilhoASerRemovido)`

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

Manipulação da Árvore DOM – Exemplo

```
<input type="text">
<button>Adicionar</button>
<ol>
  <li>Shows</li>
  <li>Filmes</li>
</ol>
<script>
  const botaoAdicionar = document.querySelector("button");
  botaoAdicionar.addEventListener("click", function () {
    const campoInteresse = document.querySelector("input");
    const listaInteresses = document.querySelector("ol");

    const novoLi = document.createElement("li");
    novoLi.textContent = campoInteresse.value;

    listaInteresses.appendChild(novoLi);
    campoInteresse.value = '';
  })
</script>
```



Quando o usuário preencher o campo e clicar no botão **Adicionar** será inserido um novo item na lista com o texto informado pelo usuário.

Manipulação da Árvore DOM – Exemplo

```
<input type="text">
<ol>
  <li>Shows</li>
  <li>Filmes</li>
</ol>
<script>
  const campoInteresse = document.querySelector("input");
  campoInteresse.addEventListener("keyup", e => {
    if (e.key === "Enter") {
      const listaInteresses = document.querySelector("ol");
      const novoLi = document.createElement("li");
      novoLi.textContent = campoInteresse.value;
      listaInteresses.appendChild(novoLi);
      campoInteresse.value = '';
    }
  });
</script>
```

Este exemplo é apenas uma alteração do exemplo anterior para permitir a inserção do novo item na lista quando o usuário pressionar a tecla Enter depois de digitar o novo interesse (não há o botão Adicionar). Repare que a função tratadora do evento tem um parâmetro de nome e contendo informações sobre o evento (no exemplo precisamos acessar a propriedade key desse objeto para verificar qual tecla foi pressionada).

Manipulação da Árvore DOM – Exemplo

```
const campoInteresse = document.querySelector("input");
campoInteresse.addEventListener("keyup", e => {
  if (e.key === "Enter") {

    const novoLi = document.createElement("li");
    const novoSpan = document.createElement("span");
    const novoBotao = document.createElement("button");

    novoSpan.textContent = campoInteresse.value;
    novoBotao.textContent = '✕';

    novoLi.appendChild(novoSpan);
    novoLi.appendChild(novoBotao);
    const listaInteresses = document.querySelector("ol");
    listaInteresses.appendChild(novoLi);

    novoBotao.onclick = function () {
      listaInteresses.removeChild(novoLi);
      // Outra opção: novoLi.remove();
      // Outra opção: novoLi.parentNode.removeChild(novoLi);
    }
    campoInteresse.value = '';
  }
})
```

Este exemplo é uma alteração do exemplo anterior que adiciona um botão "x" para permitir ao usuário excluir o item da lista posteriormente. Observe que cada item de lista passa a conter dois elementos, um para o texto e um <button> para o botão "x". Repare também que foi adicionada uma função para tratar o evento click no botão "x".

Conteúdo Complementar

Outras Propriedades e Métodos para Manipulação da Árvore DOM

node.parentNode

- retorna o nó pai do nó em questão

node.children

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

node.nextElementSibling

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

node.previousElementSibling

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

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>
```

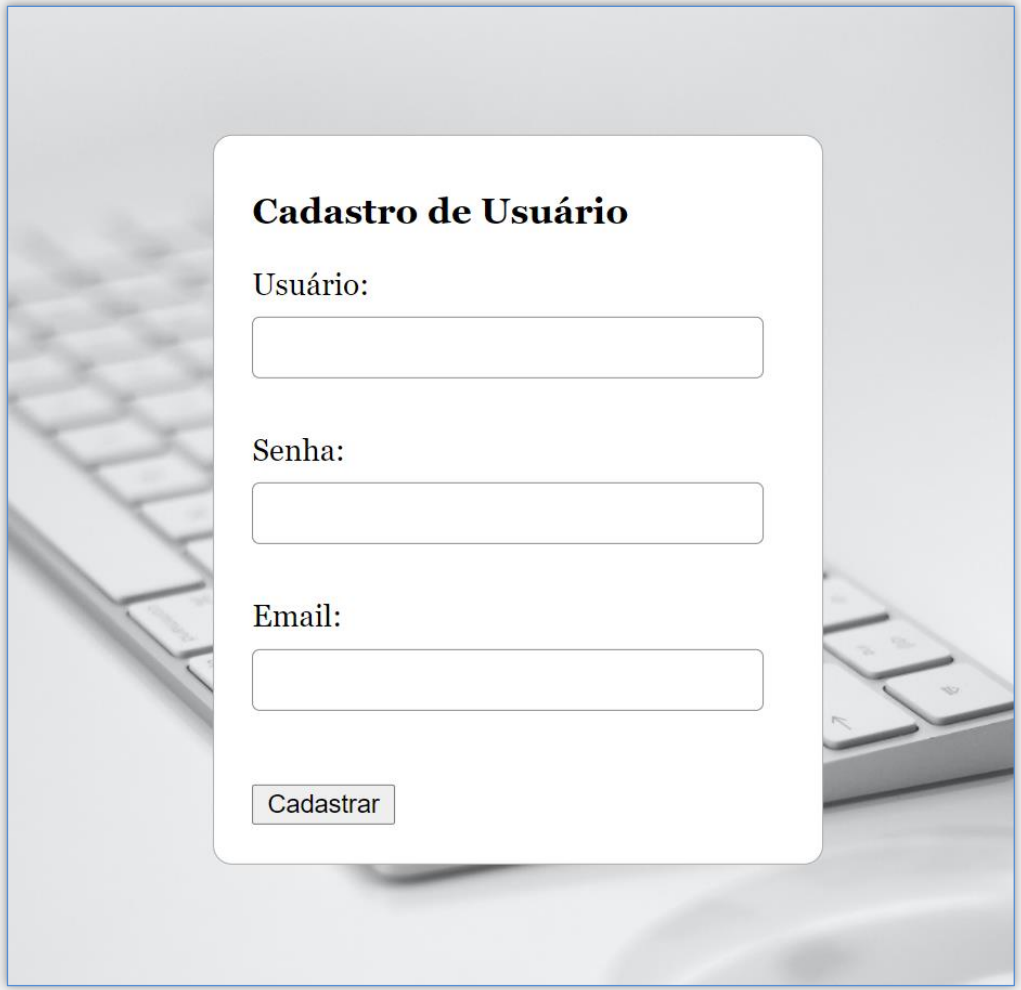
```
const campoProduto = document.forms.cadastro.produto;
const valorDoCampo = campoProduto.value;
const ultNome = document.forms.cadastro["ultimo-nome"].value;
```

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 – HTML

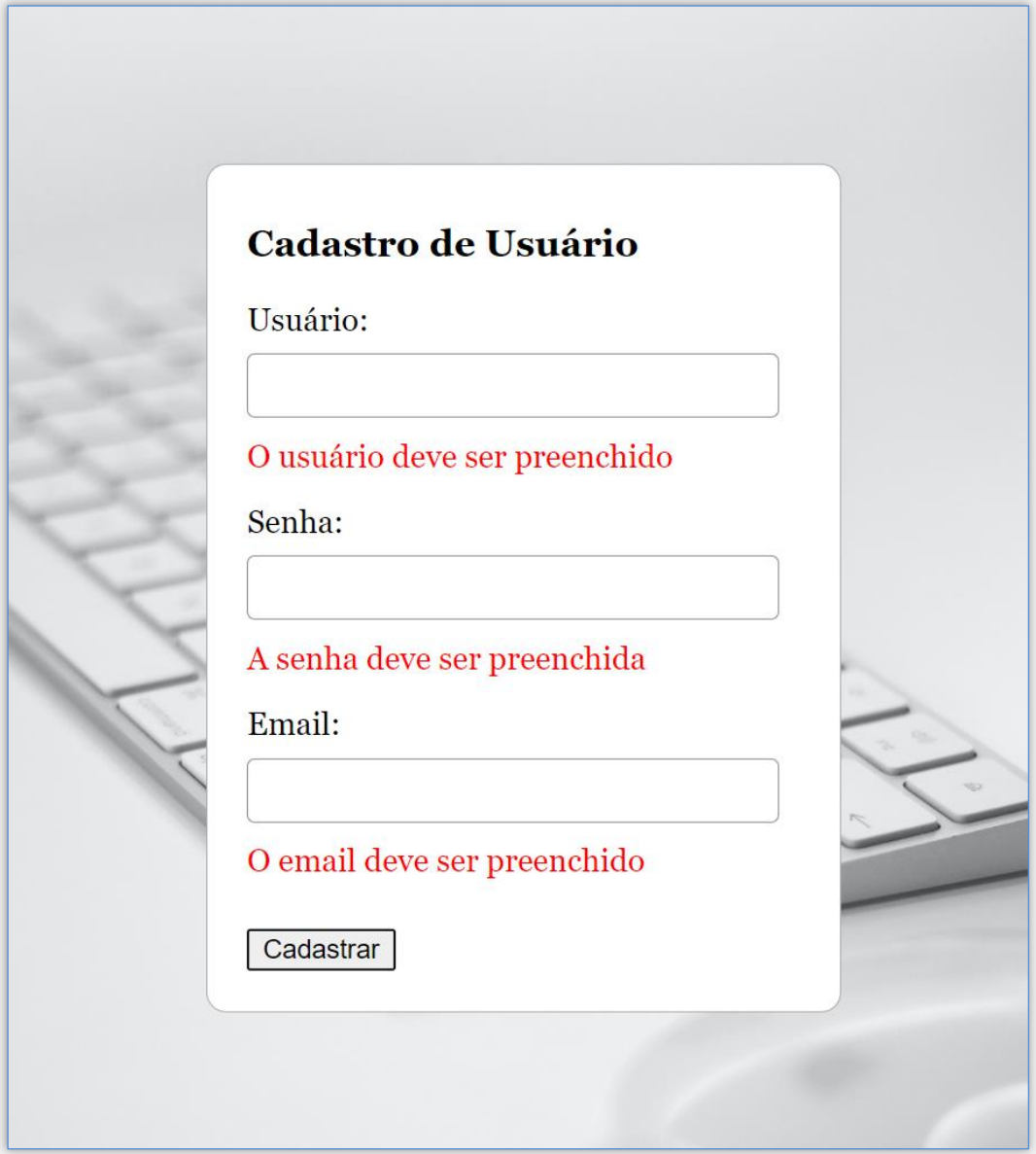
```html
<form name="cadastro" action="login.php" method="post">
  <div>
    <label for="usuario">Usuário:</label>
    <input type="text" id="usuario" name="usuario">
    <span></span>
  </div>
  <div>
    <label for="senha">Senha:</label>
    <input type="password" id="senha" name="senha">
    <span></span>
  </div>
  <div>
    <label for="email">Email:</label>
    <input type="email" id="email" name="email">
    <span></span>
  </div>
  <button>Cadastrar</button>
</form>
```



Validação de Formulário – JavaScript

```
document.forms.cadastro.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 = '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.



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**.