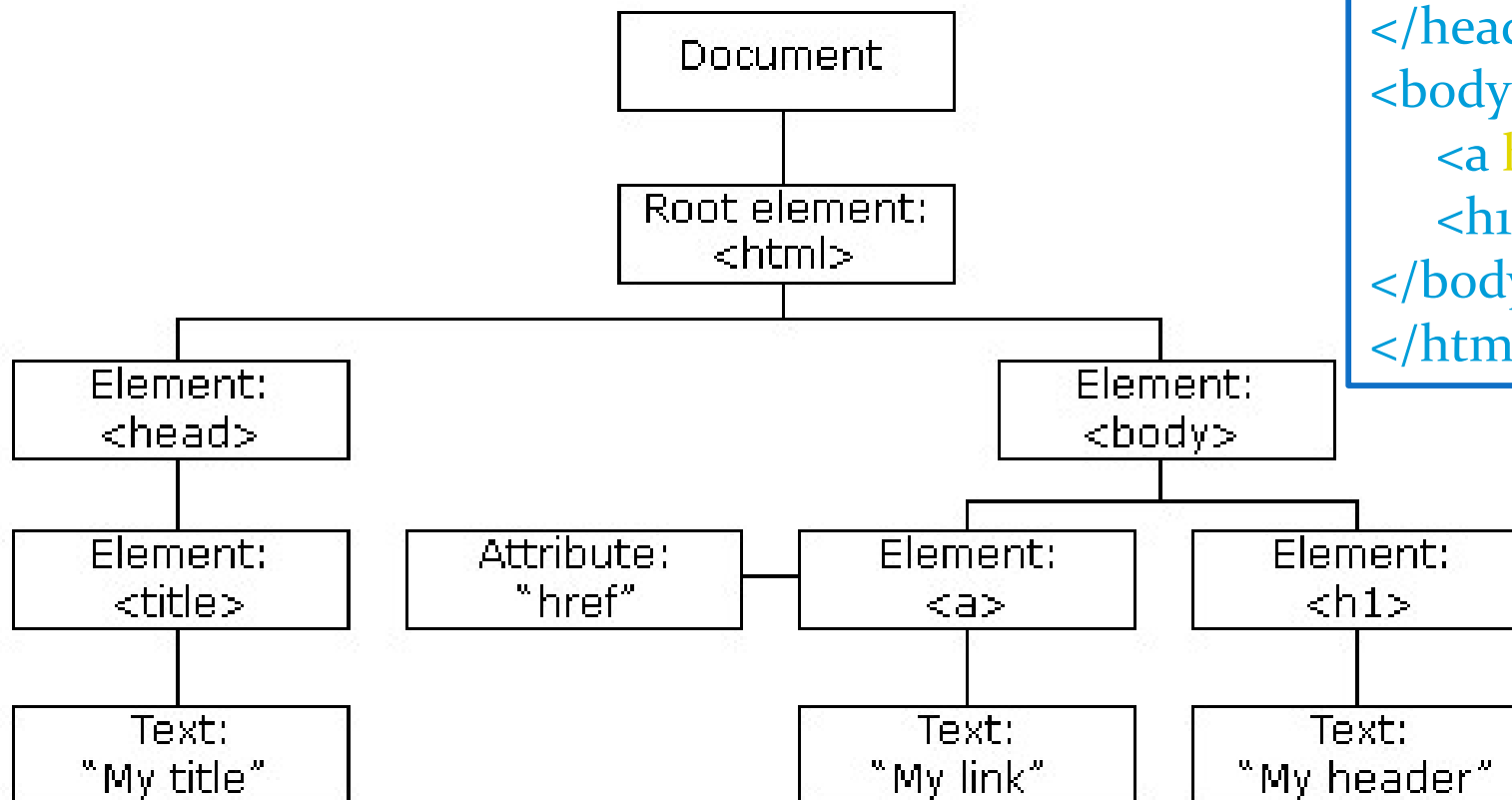


Introdução ao JavaScript – parte3

Profª Mª Denilce Veloso
denilce.veloso@fatec.sp.gov.br
denilce@gmail.com

HTML DOM (Document Object Model)

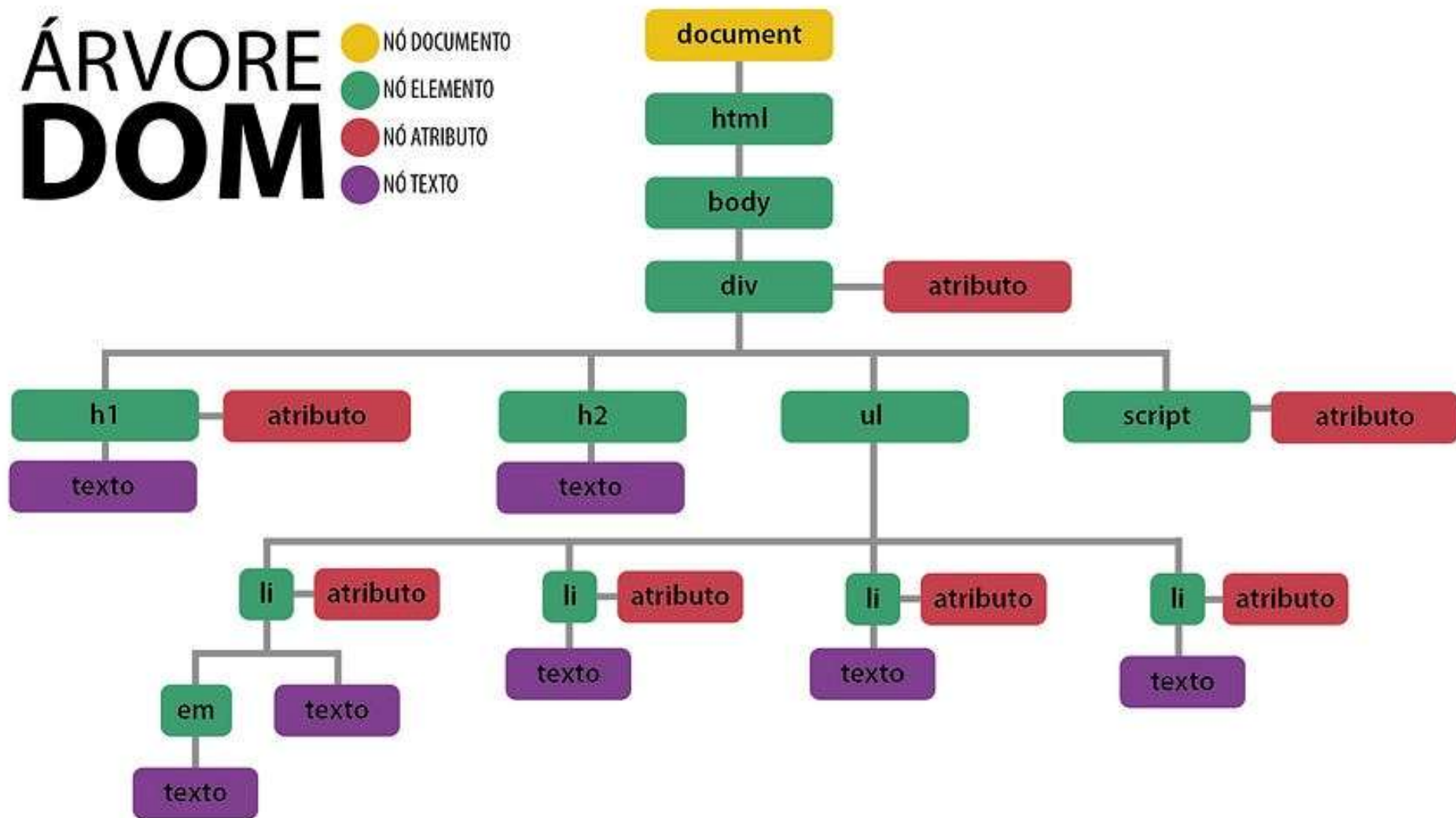
Quando uma página é lida no navegador ela torna-se um object Document.



```
<html>
<head>
<title>My title</title>
</head>
<body>
  <a href="">My link</a>
  <h1>My header</h1>
</body>
</html>
```

ÁRVORE DOM

- NÓ DOCUMENTO
- NÓ ELEMENTO
- NÓ ATRIBUTO
- NÓ TEXTO



Fonte: https://miro.medium.com/v2/resize:fit:828/format:webp/1*mMmuOhNytgqP7lrU9HPTpw.jpeg

HTML DOM (Document Object Model)

Tabela de Node Type

Node type		nodeName returns	nodeValue returns
1	Element	element name	null
2	Attr	attribute name	attribute value
3	Text	#text	content of node
8	Comment	Represents a comment	None

https://www.w3schools.com/jsref/prop_node_nodetype.asp

HTML DOM - Nós

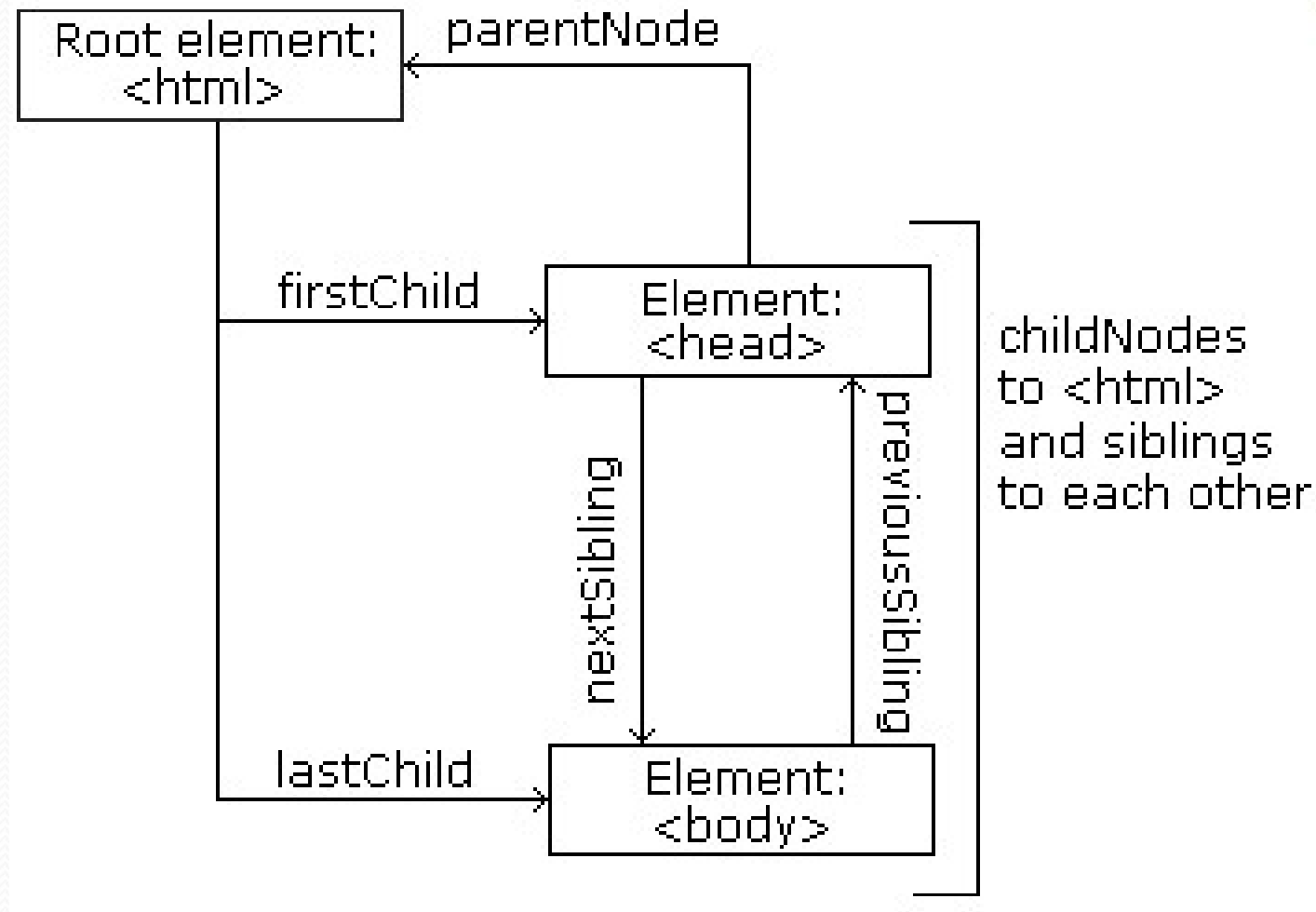
No HTML DOM tudo é um nó.

- ✓ O documento propriamente dito é um nó documento;
- ✓ Todos os elementos HTML são nós de elemento;
- ✓ Todos os atributos HTML são nós de atributo;
- ✓ O texto dentro dos elementos HTML são nós de texto;
- ✓ Os comentários são nós de comentário.

HTML DOM – permite:

- ✓ Encontrar e definir os valores dos elementos em uma página;
- ✓ Manipulação de eventos para os controles em uma página;
- ✓ Modificar os estilos associados aos elementos;
- ✓ Validar e atualizar páginas da web;
- ✓ Etc.

HTML DOM – Relacionamento entre nós



Sibling (“irmão”) – imediatamente anterior ou próximo

HTML DOM – Relacionamento entre nós

```
<html>

  <head>
    <title>Exemplo DOM</title>
  </head>

  <body>
    <h1>Lição Um</h1>
    <p>Alô Mundo!</p>
  </body>

</html>
```

Nesse exemplo:

<html> é nó raiz

<html> não tem “parents” ou pais

<html> é parent de <head> e <body>

<head> é o “first child” primeiro filho de <html>

<body> é o “last child” último filho de <html>

<head> e <body> são siblings (irmãos)

<head> tem um “child” filho <title>

<title> tem um filho (text node - texto): “Exemplo DOM “

<body> tem dois filhos <h1> e <p>

<h1> tem um filho (texto) “Lição Um”

<p> tem um filho (texto) “Alô Mundo!”

<h1> e <p> são siblings (irmãos)

HTML DOM – Propriedades

Supondo x (nó) o elemento, atributo ou texto:

- ✓ x.innerHTML: o valor text de x (**inclui** as tags e textos);
- ✓ x.innerText: o valor text de x (**exclui** as tags);
- ✓ x.nodeName: o nome do elemento x; Por ex.: INPUT
- ✓ x.nodeValue: o valor de x; (por exemplo: do Input)
- ✓ x.nodeType: o tipo de x (1 – elemento; 2 – atributo; 3 – texto);
- ✓ x.parentNode: o nó pai de x;
- ✓ x.childNodes: os nós filhos de x;
- ✓ x.attributes: os atributos de x;
- ✓ x.getAttribute("atributo"): o atributo do elemento. Por ex. name, id, src, class, value etc

HTML DOM - Métodos

Supondo x (nó) o elemento, atributo ou texto:

- ✓ x.getElement**ById**(id): obtém o elemento com o id fornecido;
- ✓ x.getElements**ByTagName**(name): obtém todos os elementos com a tag name;
- ✓ x.getElements**ByClassName**(name): obtém todos os elementos com a class name;
- ✓ x.**appendChild**(node): insere um nó filho node em x;
- ✓ x.**removeChild**(node): remove o nó filho node de x;

HTML DOM - Métodos

- ✓ x.**querySelector**(seletor): procura pelo primeiro elemento na sua página que corresponde a um determinado critério (um seletor CSS). Se nenhum elemento for encontrado, ele retorna null. Por exemplo: seletor de tag, seletor de classe, seletor de ID etc.

```
const primeiroParagrafo = document.querySelector('p')  
const primeiroBotao = document.querySelector('.botao');  
const cabecalho = document.querySelector('#cabecalho');
```


HTML DOM - Métodos

- ✓ x.**querySelectorAll**(seletor): retorna uma coleção de todos os elementos correspondentes, na forma de um **NodeList**. Essa NodeList se parece com um array, mas não é exatamente um array. É possível iterar sobre ela usando o método `forEach`.

```
const itensDaLista = document.querySelectorAll('ul li');
```

```
const matches = document.querySelectorAll("div.nota, div.alerta");
```

```
const todosOsItens = document.querySelectorAll('.item');
```

- ** ver UsandoQuerySelector.html

HTML DOM – Outras Propriedades e Métodos

✓ `hasFocus()` – Método retorna true se o item possui o foco;

```
let meuInput = document.getElementById("meuInput");  
if (meuInput.hasFocus()) {  
  console.log("O input tem o foco");  
}
```

✓ `activeElement` – Contém o elemento que possui o foco no momento.

```
const activeEl = document.activeElement;  
console.log(activeEl.tagName); // retorna a tag (ex. INPUT) do elemento  
atualmente ativo  
console.log("Elemento com foco:", activeEl.id); // retorna o id do elemento ativo
```



```
<!DOCTYPE html>
<html lang="pt-br">
<head>
</head>
<body>
  <script>
    function executa() {
      paragrafo = document.getElementById("primeiro");
      paragrafo.innerHTML = "<strong>" + " Texto do parágrafo primeiro: " +
        paragrafo.innerHTML + "</strong>"
      let x = document.getElementsByName("check");
      for (i = 0; i < x.length; i++) {
        if (x[i].type == "checkbox") {
          x[i].checked = true; } } }
  </script>
  <p id="primeiro"> Boa Noite Turma 1 </p>
  <input type="text" name="texto1" value="AAA" size=20>
  <br> <br>
  ESCOLHA1: <input type="checkbox" name="check">
  ESCOLHA2: <input type="checkbox" name="check">
  <button onclick="return executa();"> clique aqui </button>
</body>
</html>
```


Slide 14

DV0

Embora seja possível utilizar eventos diretamente no elementos, as práticas modernas de JavaScript favorecem o uso de `addEventListener` para tratamento de eventos.

DENILCE DE ALMEIDA OLIVEIRA VEL; 2025-04-25T19:45:35.924

HTML DOM – Exemplo 2 (1)

Ver exemplo UsandoDOM2Nomes.html – como acessar os valores dos campos

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

```
<head>
```

```
  <meta charset="UTF-8">
```

```
  <title>Teste DOM</title>
```

```
  <script>
```

```
    function validarDados() {
```

```
      // usando a posicao no formulario
```

```
      if (document.forms.formulario1.elements[o].value == "" ||
```

```
document.forms.formulario1.elements[o].value.length < 3) {
```

```
        alert("Preencha campo NOME corretamente!");
```

```
        document.forms.formulario1.elements[o].focus();
```

```
        return false;
```

```
      };
```


HTML DOM – Exemplo 2 (2)

Ver exemplo UsandoDOM2Nomes.html – como acessar os valores dos campos

// usando o nome no formulario

```
if (document.forms.formulario1.elements["idEmail"].value == "" ||  
document.forms.formulario1.elements["idEmail"].value.indexOf('@') == -1 ||  
document.forms.formulario1.elements["idEmail"].value.indexOf('.') == -1) {  
    alert("Preencha campo E-MAIL corretamente!");  
    document.forms.formulario1.elements.idEmail.focus();  
    return false;  
}
```

//usando o nome no formulario de maneira direta

```
if (document.forms.formulario1.elements.idMensagem.value == "" ||  
document.forms.formulario1.elements.idMensagem.value.length < 50) {  
    alert("É necessario preencher o campo MENSAGEM com mais de 50  
caracteres!");  
    document.getElementById("idMensagem").focus();  
    return false;  
}
```

return true;

}

</script>

HTML DOM – Exemplo 2 (3)

Ver exemplo UsandoDOM2Nomes.html – como acessar os valores dos campos

```
</head>
```

```
<body>
```

```
  <form action="teste" method="POST" name="formulario1" id="formulario1"
onsubmit="return validarDados();">
```

```
    <div id="principal" width=200 height=200 align=center>
```

```
      Nome Completo:
```

```
      <input name="idNome" type="text" id="idNome" size="52"
maxlength="150">
```

```
      <br>
```

```
      <br> E-Mail :
```

```
      <input name="idEmail" type="email" id="idEmail" size="52"
maxlength="150">
```

```
      <br>
```

```
      <br> Mensagem :
```

```
      <textarea name="idMensagem" cols="50" rows="8" id="idMensagem"
value="" input></textarea>
```


HTML DOM – Exemplo 2 (4)

Ver exemplo UsandoDOM2Nomes.html – como acessar os valores dos campos

```
<br>
    <br>
    <br>
    <input name="Submit" type="submit" form="formulario1"
value="Submit">
    <input name="Reset" type="reset" form="formulario1"
value="Redefinir">
    </div>
</form>
</body>

</html>
```


HTML DOM – Exemplo 3 (1)

Ver exemplo UsandoDOM3Nos.html – Trabalhando com os Nós.

```
<!DOCTYPE html>
<head>
  <title>Teste DOM</title>
  <script src="UsandoDOM3Nos.js"></script>
</head>
<body id="principal">
  <ul id="Lista" name="Lista">
    <li>Item1</li>
  </ul>

  <p id="idp1" name="namep1">
    <b>Este é um teste de parágrafo</b>
  </p>
  <br> <br>
  <button onclick="adicionaItem('Item2')">Adiciona item2 lista</button>
  <button onclick="adicionaItem('Item3')">Adiciona item3 lista</button>
  <button onclick="mostraNo('idp1')"> Mostra nó parágrafo</button>
</body>

</html>
```


HTML DOM – Exemplo 3 (2) - UsandoDOM3Nos.js

```
// adiciona item no nó  
function adicionalItem(nome) {  
    document.getElementById('Lista').innerHTML += "<li>" + nome + "</li>";  
};
```

```
// mostra os dados do nó  
function mostraNo(item) {
```

```
    alert("Valor innerHTML do principal(Lista)" +  
document.getElementById(item).innerHTML);
```

```
    alert("Nome da tag Nó " + document.getElementById(item).nodeName);  
    alert("Name do Nó +  
document.getElementById(item).getAttribute("name"));  
    // se fosse button por exemplo o value seria o texto, input o texto digitado  
    // no caso parágrafo não tem value  
    alert("value do Nó " + document.getElementById(item).nodeValue);
```


HTML DOM – Exemplo 3 (3) – UsandoDOM3Nos.js

```
//pai
alert("Parent " +
document.getElementById(item).parentNode.nodeName);

// o primeiro filho de elemento parágrafo é tipo texto
alert("Primeiro Filho " +
document.getElementById(item).firstChild.innerHTML + " type " +
document.getElementById(item).firstChild.nodeType);
document.getElementById(item).firstChild.nodeValue;
};
```


Slide 21

DV0

Se `childNodes[0]` está retornando `undefined`, isso geralmente ocorre porque o primeiro filho do elemento `<p>` é considerado um nó de texto vazio devido aos espaços em branco e quebras de linha no código HTML. Portanto, o primeiro filho real (o elemento `<bold>`) estaria na posição 1 em `childNodes`.

DENILCE DE ALMEIDA OLIVEIRA VEL; 2024-05-05T14:05:24.843

HTML DOM – Exemplo 4 - Ver exemplo

UsandoDOM2Select.html

```
<body>
  <select name="cidades" id="cidades" onchange="testar();">
    <option value="sorocaba">Sorocaba</option>
    <option value="votorantim">Votorantim</option>
    <option value="itu">Itu</option>
    <option value="aracoiaba">Araçoiaba da Serra</option>
  </select>
  <script>
    function testar() {
      let selecao = document.getElementById("cidades");
      let opcaoTexto = selecao.options[selecao.selectedIndex].text;
      let opcaoValor = selecao.options[selecao.selectedIndex].value;

      alert(` texto: ${opcaoTexto} \n valor: ${opcaoValor} `);
    }
  </script>
</body>
```


HTML – DOM – Métodos

SetAttribute e GetAttribute

setAttribute - define o valor de um atributo em um elemento. Passar atributo e valor.

Exemplo:

```
let image = document.getElementById("myImage");  
image.setAttribute("src", "imagem.jpg");
```

getAttribute - é usado para obter o valor de um atributo de um elemento. Passar atributo.

Exemplo:

```
let image = document.getElementById("myImage");  
let srcValue = image.getAttribute("src");  
console.log(srcValue);
```


PADRÕES NO JAVASCRIPT

Para JS

<https://gist.github.com/vinicius-stutz/1b37cb84b6240efe6ab8137660a15640#:~:text=Segundo%20Douglas%20Crockford%2C%20todos%20os,como%20o%20C%23%2C%20por%20exemplo.>

<http://crockford.com/javascript/>

Para CSS

<https://desenvolvimentoparaweb.com/css/bem/>
<http://www.andrefelizardo.com.br/blog/padroes-css/#:~:text=trumps-,Padr%C3%B5es%20CSS%20de%20nomenclatura,CSS%20escrita%20para%20cada%20op%C3%A7%C3%A3o.>

ou

<https://wbruno.com.br/css/afinal-como-nomear-ids-e-classes-no-csshtml/>

HTML DOM – Exercício



Qual será o resultado na tela?

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

```
<script>
```

```
  let i = 1 + 2 + "3";
```

```
  for (j = 0; j < 10; j++) {
```

```
    if (j === 5) {
```

```
      continue;
```

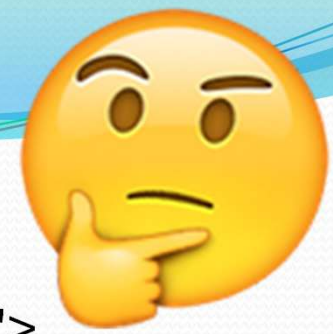
```
    }
```

```
    document.getElementById("teste").innerHTML += i + j + "<br>";
```

```
  }
```

```
</script>
```


HTML DOM – Exercício



DV0

```
<form name="formulario1">  
  <input name="idNome" type="text" id="idNome" value="teste">  
</form>
```

Quais alerts irão mostrar o valor que for digitado no Input?

```
alert("1 " + document.formulario1.elements[0].value);  
alert("2 " + document.forms.formulario1.elements.idNome.value);  
alert("3 " + document.forms.formulario1.elements[0].value);  
alert("4 " + document.getElementById("idNome"));  
alert("5 " + document.forms.formulario1.elements["idNome"].value);  
alert("6 " + document.getElementById("idNome").value);  
alert("7 " + document.getElementsByTagName("idNome"));  
alert("8 " + document.forms.getElementByName("idNome"));  
alert("9 " + document.getElementByName("idNome").value);  
alert("10 " + document.getAttribute("name"));
```

- a) Todos
- b) Somente 1,2,3,4,5,6
- c) Somente 1,2,3,5,6
- d) Somente 1,2,3,4,5,6,7,8,10
- e) Somente 1,2,3,4,5,6,9,10
- f) Nenhuma das alternativas

DV0

Letra c

DENILCE DE ALMEIDA OLIVEIRA VEL; 2024-09-04T14:15:30.934

Exercício

- ✓ Criar um Input Text e dois Input RadioButton.
 - 1º RadioButton - transforma o conteúdo do Input Text em letras maiúsculas.
 - 2º RadioButton - transforma o conteúdo do Input Text em letras minúsculas.

Dica: *Exemplo UsandoDOM2Nomes.html* mostra exemplos de como acessar os valores dos campos.

Disponibilizar como Atividade14 no GITHUB.

→ Seuusuario/PWEB/Atividade14

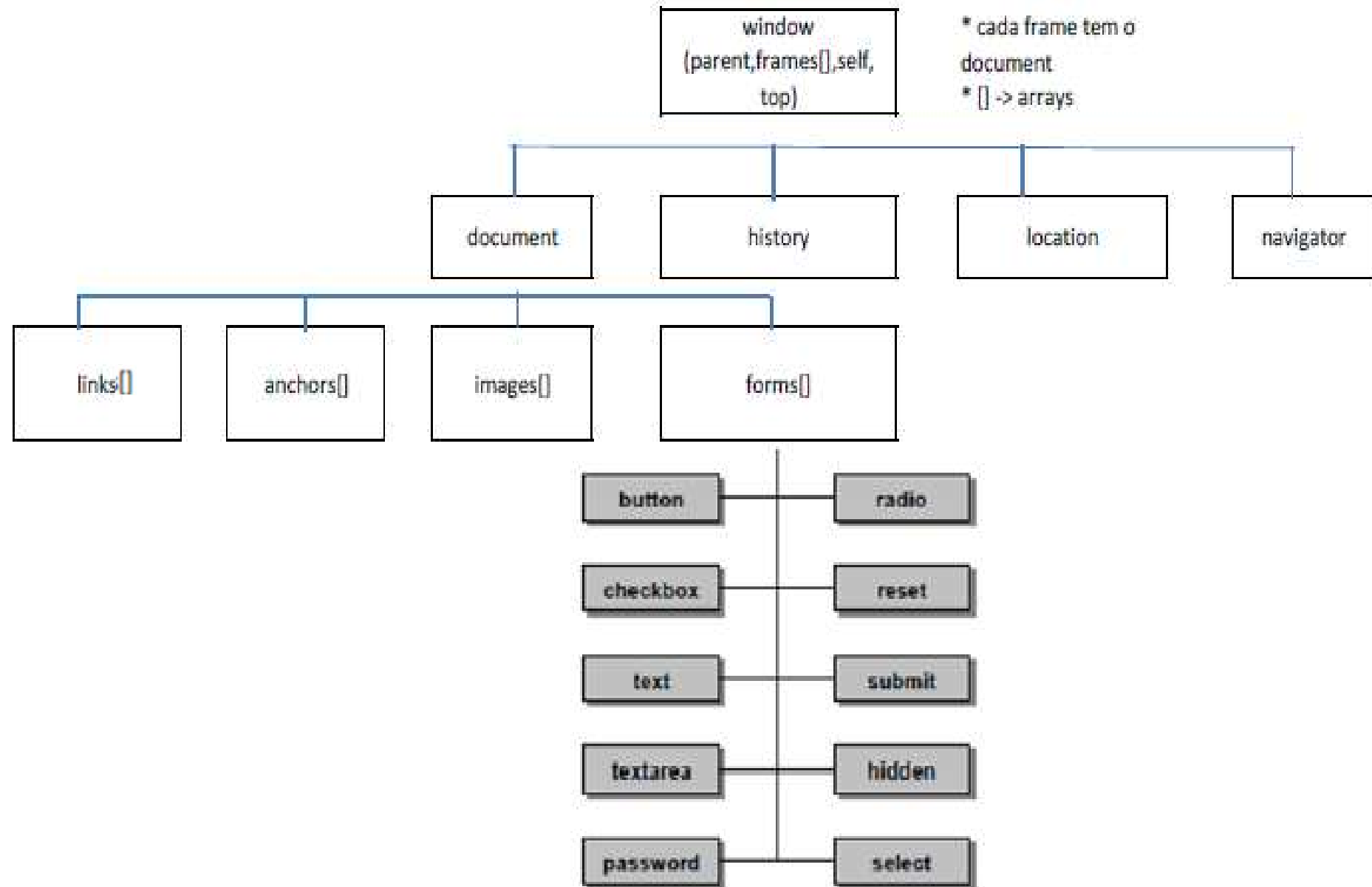
JavaScript – Outros Objetos HTML DOM

- ✓ Para trabalhar com o navegador e os documentos, o JavaScript utiliza objetos browser.
- ✓ Esses objetos estão organizados seguindo uma hierarquia: objeto pai seguido pelo nome ou nomes do objeto filho (separado por pontos).

Exemplo: window.document.imagem1

*** Por convenção, para acessar um objeto, é necessário indicar todo o caminho na hierarquia. Porém, ao escrever document.imagem1, supõe-se estar se referindo a um objeto da janela em uso. Usar caminho completo quando estiver trabalhando com várias janelas.*

JavaScript – Hierarquia objetos DOM



JavaScript – window

- ✓ Window: objeto no topo da hierarquia, representa a janela do navegador, refere-se sempre à janela atual (self).

DV0

Propriedades:

- Self – se refere a própria janela
- Top sempre se refere à janela superior na hierarquia de janelas.
- Parent se refere à janela pai da janela atual, independentemente da posição na hierarquia de janelas (é a mais próxima).

DV0

Nesse caso, a propriedade top se refere à janela superior na hierarquia de janelas. Isso significa que se você estiver na janela filho, a propriedade top apontará para a janela pai que a abriu. Por outro lado, se você estiver na janela pai, a propriedade top apontará para a própria janela pai.

Já a propriedade parent se refere sempre à janela pai da janela atual. Isso significa que se você estiver na janela filho, a propriedade parent apontará para a janela pai que a abriu. Se você estiver na janela pai, a propriedade parent também apontará para a própria janela pai.

DENILCE DE ALMEIDA OLIVEIRA VEL; 2023-05-10T16:10:21.937

JavaScript – Objeto Window – Métodos

✓ *window.alert()*

✓ *window.prompt()*

✓ *window.confirm()*

*** Não precisa usar o nome window antes dos métodos quando estiver com apenas uma janela aberta.*

JavaScript – Objeto Window – Propriedades e Métodos

- ✓ `window.setTimeout()` - Permite a execução de comandos com temporizador.
- ✓ `window.clearTimeout()` - Interrompe a execução de um temporizador antes do tempo marcado.

EX.:

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8" />
    <title>Objetos DOM</title>
    <script>
        n = 0;

        function atualizar() {
            n++;
            document.getElementById("contador").innerHTML = "<p>" + n + "</p>";
            temp1 = window.setTimeout("atualizar();", 1000);
        }
    </script>
</head>
<body>
    <br><br>
    <span id="contador"></span>

    <a href="#" onClick="window.clearTimeout(temp1);"> Parar o contador
</a>

    <script>
        atualizar();
    </script>

</body>
</html>
```

****UsandoDom7SetTimeOut.html**

JavaScript – Objeto Window – Propriedades e Métodos

- ✓ *window.open()* - Abre uma nova janela.
- ✓ *window.close()* – Fecha a janela.
- ✓ *window.print()* – Imprime a página.
- ✓ *window.navigator* - Esta propriedade contém informações sobre o navegador, nome, versão, e outras informações. Possui o método:

JavaScript – Objeto Window – Exemplo: open, close, print

```
<!DOCTYPE html>
```

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

```
<head>
```

```
  <meta charset="UTF-8" />
```

```
  <title>Objetos DOM</title>
```

```
<script>
```

```
  function abreJanela() {
```

```
    // nome janela, url, propriedade name, medidas
```

```
    novaJanela = window.open("pagina1.html", "janela1", "width = 400, height = 400");
```

```
  }
```

```
</script>
```

```
</head>
```

```
<body>
```

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

```
  <script>
```

```
    abreJanela();
```

```
    document.getElementById("teste").innerHTML = "nome:" + window.navigator.appName + "  
    versão:" + window.navigator.appVersion;
```

```
  </script>
```

```
  <input type="button" value="Fecha Nova Janela" onclick="novaJanela.close();">
```

```
  <input type="button" value="Imprime esta página" onclick="!print();">
```

```
</body>
```

```
</html>
```

** Usando Dom8OpenClose.html

Não deixar
bloqueador pop-
ups ativado

JavaScript – Objeto window.load - Exemplo

```
!DOCTYPE html>
```

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

```
**UsandoDom5Windowload.html
```

```
<head>
```

```
  <meta charset="UTF-8" />
```

```
  <title>Vendas – Window </title>
```

```
  <style>
```

```
    .coron {
```

```
      background: red;
```

```
    };
```

```
</style>
```

```
<script>
```

```
  window.onload = function () { // load ocorre no carregamento do objeto
```

```
    // +=2 para pular uma linha e ficar listrado
```

```
    let colorir = document.querySelectorAll('tbody tr');
```

```
    for (var i = 0; i < colorir.length; i += 2)
```

```
      colorir[i].className = 'coron'; // atribui nome classe
```

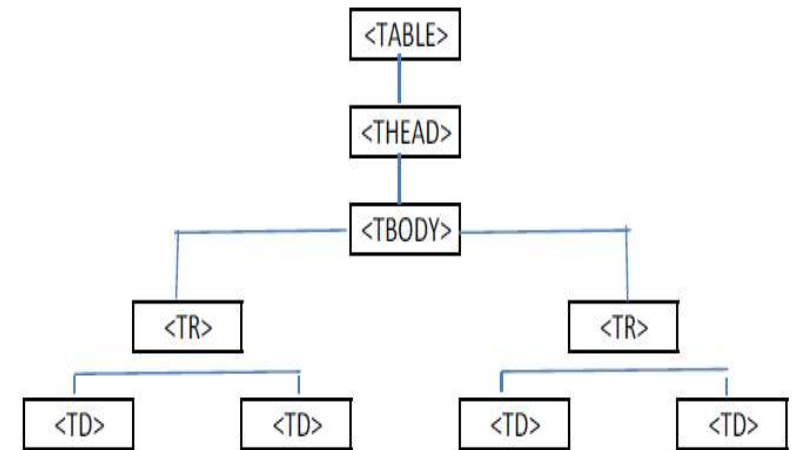
```
  }
```

```
</script>
```

```
</head>
```


JavaScript – Objeto window.load - Exemplo

```
<body>
  <table class="venda">
    <thead>
      <tr> <th>Vendedor</th> <th>Total</th>
      </tr> → DENTRO DE UMA LINHA, 2 CABECALHOS
    </thead>
    <tbody>
      <tr> <td>Ana</td> <td>10.000,00</td> → TD - CÉLULA
      </tr>
      <tr><td>Pedro</td><td>50.000,00</td>
      </tr>
      <tr><td>Maria</td><td>13.200,00</td>
      </tr>
      <tr><td>Celeste</td><td>44.999,00</td>
      </tr>
      <tr><td>Livia</td><td>45.780,00</td>
      </tr>
      <tr><td>Medson</td><td>78.000,00</td>
      </tr>
    </tbody>
  </table>
</body>
</html>
```



JavaScript – Frames

Frames **divisões da janela do navegador em múltiplos quadros/painéis.** Cada frame pode conter uma página diferente ou a saída de um script. Cada frame é considerado um objeto equivalente ao objeto window (tem os outros objetos, propriedades que o window tem).

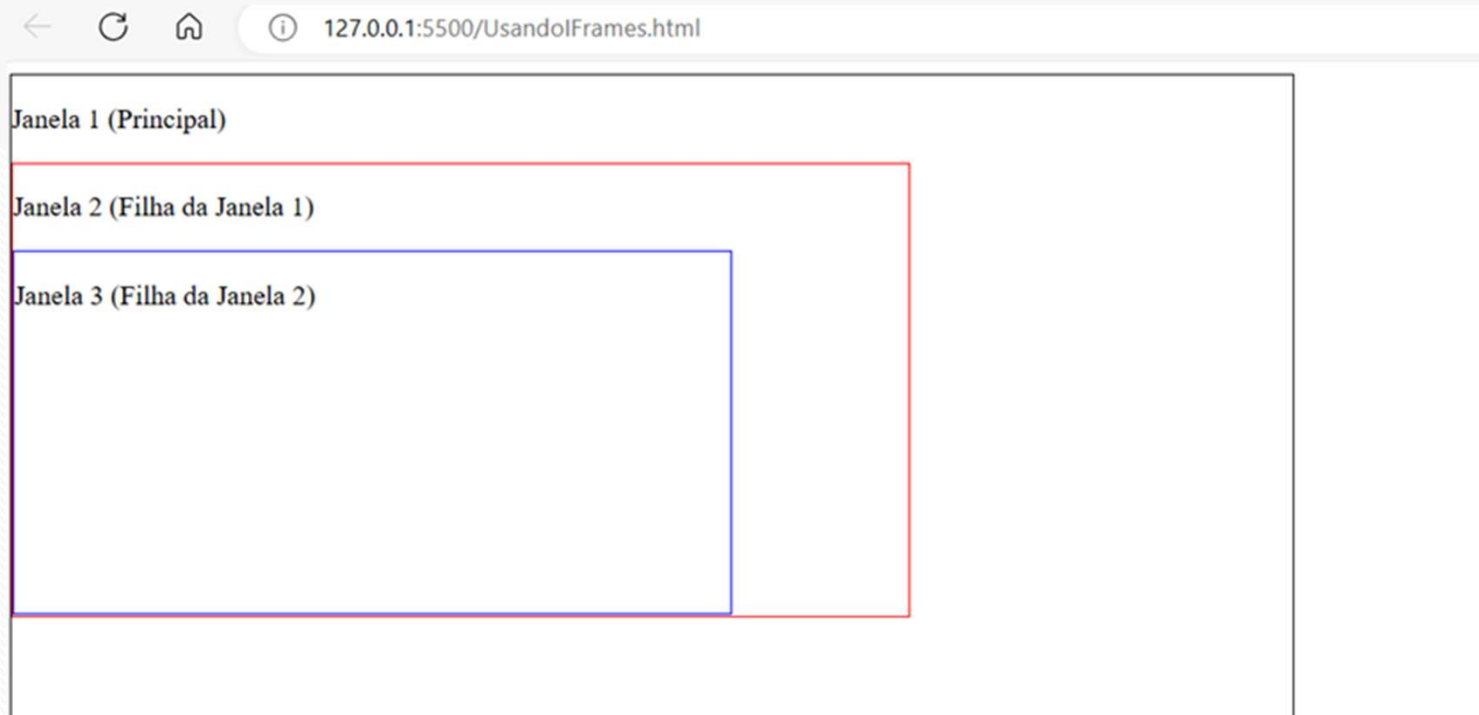
JavaScript – Window Frames - Array

É possível referenciar os frames pelo índice, a partir de 0 (zero). No exemplo ANTERIOR `parent.frames[0]` é equivalente ao frame “esquerdo” e `parent.frames[1]` é o frame “direito”.

Exemplo usando linhas e colunas: [UsandoDOM11Frames3.html](#)

Uso de Iframe (HTML 5)

- O **<iframe>** é uma tag HTML que permite incorporar outro documento HTML dentro de uma página.
- Pode ser usado para exibir conteúdo de outra página da web em um quadro dentro da página principal.
- Cada **<iframe>** é um elemento independente, e você pode ter vários **<iframe>** na mesma página.



**** Ver exemplo UsandoIFrames.html**

JavaScript – history

History: contém histórico das URLs visitadas.

Propriedades:

- ✓ *window.history.length* - length armazena a quantidade de localizações diferentes visitadas.
- ✓ *window.history.current* - current, assim como a *window.location.href*, contém o endereço da página atual (só que da página que está no histórico).
- ✓ *window.history.next* - next contém o endereço da próxima página (para onde o usuário foi e depois retornou), podendo recarregá-la novamente através do botão avançar.
- ✓ *window.history.previous* - previous armazena o endereço da página anterior (de onde o usuário veio), podendo retornar através do botão retornar.

JavaScript – history

Métodos:

✓ *window.history.go()* - *go()* permite a navegação entre as páginas já visitadas. Argumento com valor positivo ex. *go(1)* avança para a próxima página já visitada. (Next)

Argumento com valor negativo ex. *go(-1)* retorna para a página anterior visitada. (Back).

✓ *window.history.back()* - *back()* retorna à página anterior. (Back) Similar *go(-1)*

✓ *window.history.forward()* - *forward()* avança para a próxima página. (Next) Similar *go(0)*

JavaScript – location

Location: contém informação sobre a URL corrente.

Propriedades:

- ✓ *window.location.href* – href armazena o URL da página atual.
- ✓ *window.location.protocol* - protocol armazena o protocolo da página atual, basicamente http.

Métodos:

- ✓ *window.location.reload()* - reload() recarrega a página atual .
- ✓ *window.location.replace()* – *replace()* substitui a página atual por outra, mas não atualiza o histórico de navegação (não é possível voltar). Ex.:
`location.replace(" https://www.uol.com.br/");`

** UsandoDOM11Location11.html

**Exemplo sobre navegação de frames usando location: UsandoDOM10Frames2.html

JavaScript – document

Document - representa a estrutura hierárquica da página HTML carregada em uma janela ou guia do navegador.

Propriedades e Métodos:

- ✓ *window.document.URL* – URL endereço da página atual
- ✓ *window.document.title* - title armazena o título da página, que é exibido na barra de título do navegador.
- ✓ *window.document.referrer*- referrer armazena o endereço da página anterior (usuário estava visualizando anteriormente).
- ✓ *window.document.lastModified* - lastModified armazena a data da última atualização efetuada na página.
- ✓ *window.document.write()* - write imprime texto em um documento. Para imprimir um novo conteúdo, recarregar a página .
- ✓ *window.document.open()* - open utilizado para reescrever um documento primeiramente limpando o conteúdo anterior. É utilizado em novas janelas. (abrir, escrever e depois fechar)
- ✓ *window.document.close()* - close fecha o documento aberto.

**** Ver exemplo UsandoDOM12Document.html**

JavaScript – document.link

Os **links contidos em uma página** são tratados como objetos. Cada link faz parte do Array `links[]`. O endereço do primeiro link da página criado em HTML tem o índice 0.

Propriedades

```
<body onload="document.write(document.links[0]);">  
<a href="pagina1.html"> Primeira Página</a>  
</body>
```

****** refere-se à página1

- ✓ `document.links[]` ou `document.links[].href` - href armazena o endereço do link
- ✓ `document.links.length` - length armazena o número links existentes na página.

****** Ver exemplo *UsandoDOM13Links.html*

JavaScript – document.anchors

As âncoras (**links estabelecidos em qualquer parte do documento, que servem como “marcadores”**) contidas em uma página também são tratadas como objetos. Cada âncora faz parte de um Array `anchors[]`. A primeira âncora da página, criada via HTML é `document.anchors[0]`.

✓ `window.anchors[].name` - `name` armazena o nome da âncora contida na página.

✓ `window.anchors.length` - `length` armazena o número de âncoras existentes na página:

****** *Ver exemplo `UsandoDOM14Ancoras.html`*

JavaScript – document.images

As **imagens também são objetos**. As imagens do código HTML são armazenadas como elementos de um Array Images. A primeira imagem da página é `document.images[0]` .

- ✓ `window.document.images[].name` - name armazena o nome.
- ✓ `window.document.images[].border` - border armazena o valor da borda.
- ✓ `window.document.images[].complete` - complete armazena os valores true/false, indicando se a imagem já foi carregada ou não.
- ✓ `window.document.images[].height` - height armazena o valor da altura.
- ✓ `window.document.images[].width` – width armazena o valor da largura.
- ✓ `window.document.images[].hspace` - hspace armazena o valor do espaçamento horizontal da imagem .
- ✓ `window.document.images[].vspace` - hspace armazena o valor do espaçamento vertical .
- ✓ `window.document.images[].lowsrc` - lowsrc armazena o endereço da pré-imagem, carregada antes da imagem definitiva.

**** Ver exemplo UsandoDOM15Images.html**

JavaScript – document.images – Efeito RollOvers

Efeito Rollovers - Utilizando o objeto image com os eventos pode-se criar efeitos/animações com imagens. Um deles é o *Rollover*, onde uma imagem é substituída por outra quando se passa o ponteiro do mouse sobre ela.

Ex.:

```
<html>
<head>
  <title>Teste com Link</title>
</head>

<body>
  <a href="#" onmouseover="document.images[0].src='janelaaberta.png';"
onmouseout="document.images[0].src='janelafechada.png';"
onclick="document.images[0].src='janelaquebra.png';">
    
  </a>
</body>
</html>
```

** UsandoDOM15Rollovers.html

DV0

Os exemplos de manipulação de imagens (rollovers, pré-carregamento, animação) são bons para demonstrar a interação com o DOM. No entanto, o desenvolvimento web moderno geralmente usa animações CSS ou bibliotecas JavaScript para animações mais complexas

DENILCE DE ALMEIDA OLIVEIRA VEL; 2025-04-25T19:50:24.309

JavaScript – document.images – Pré-Carregamento

Pré-Carregamento de Imagens - Como as imagem demoram a ser carregadas, principalmente se forem pesadas, é possível otimizar seu carregamento, carregando-as previamente em cache antes da sua exibição.

```
<html>
<head>
  <title>Teste com Link</title>
  <script>
    imgs = new Image();
    imgs[0] = "vistaabrindo.gif";
    imgs[1] = "vistajanela.jpg";
  </script>
</head>
<body>
  <img src="" name="imagem1">
  <br>
  <img src="" name="imagem2">
  <script>
    document.images[0].src = imgs[0];
    document.imagem2.src = imgs[1];
  </script>
</body>
<html>
```

** UsandoDOM16CarregamentoImagens.html

JavaScript – document.images – Animação (1)

O objeto *Image* junto o temporizador *setTimeout()* permite a criação de animações através da exibição de uma sequência de imagens.

```
<html>
<head>
    <title>Teste Images - Animação Pernalonga</title>

    <script>
        // carrega varias imagens iniciadas por perna
        function carrega() {
            img = new Image();
            for (i = 0; i < 4; i++) {
                img[i] = "perna" + (i + 1) + ".gif";
            }
        }

        carrega();

        i = 0;
        // exibe imagens carregadas
        function exibe() {
            if (i > 3)
                i = 0;
            document.images[0].src = img[i];
            i++;
        }
    </script>
</head>
<body>
    <img alt="perna1.gif" data-bbox="33 187 953 1000"/>
</body>
</html>
```

**** UsandoDOM17Animacao01.html**

JavaScript – document.images – Animação (2)

```
// chamar novamente depois do timeout
```

```
timer = window.setTimeout("exibe();", 1000);
```

```
}
```

```
function para(){
```

```
    window.clearTimeout(timer);
```

```
}
```

```
</script>
```

```
</head>
```

```
<body bgcolor="brown">
```

```
    
```

```
    <br>
```

```
    <input type="button" value="Iniciar" size=20 onclick="exibe();">
```

```
    <input type="button" value="Parar" size=30 onclick="para();">
```

```
</body>
```

```
</html>
```


JavaScript – Animação usando animation

*** Ver exemplo UsandoDOM18Animacao2.html*

Quer saber mais sobre animação no JavaScript ?

<http://javascript.info/tutorial/animation>

<https://code.tutsplus.com/pt/tutorials/20-best-css-animations--cms-27561>

JavaScript – document.forms

Forms - uma coleção que representa todos os formulários.

Ex.:

```
<form      name="myForm"      method="post"      encoding="multipart/form-data"  
action="processa_formulario.php">
```

Propriedades

- ✓ *forms[].name* - Armazena o conteúdo de name (nome)
- ✓ *forms[].action* - Armazena o conteúdo de action. (*para onde será enviado, formulário, e-mail, etc*)
- ✓ *forms[].method* - Armazena o conteúdo de method. (*get, post*)
- ✓ *forms[].target* - Armazena o conteúdo de target (*onde ocorrerá o retorno, nova página, novo frame, etc*)
- ✓ *forms[].encoding* - Armazena tipos de codificação utilizada quando o formulário é enviado para o servidor. Por ex. `encoding="multipart/form-data"` - usada para envio de arquivos binários como imagens, documentos etc
- ✓ *forms[].length* - Armazena a quantidade de elementos (campos) (leitura)

Métodos

- ✓ *forms[].submit()* - Envia o formulário indicado
- ✓ *forms[].reset()* - Limpa o formulário indicado

application/x-www-form-urlencoded: Este é o tipo de codificação padrão para formulários HTML e é usado para enviar dados de formulário como uma string de consulta no formato URL-safe. Os caracteres especiais são codificados em "%XX", onde "XX" é o valor hexadecimal do caractere.

multipart/form-data: Este tipo de codificação é usado para enviar dados binários ou arquivos, juntamente com outros dados de formulário. Os dados são divididos em várias partes, cada uma com um cabeçalho que indica o tipo de conteúdo da parte.

text/plain: Este tipo de codificação é usado para enviar dados de texto simples sem formatação. É menos comum em formulários HTML, mas pode ser útil em certas situações, como ao enviar dados para um servidor que não é compatível com outras codificações.

application/json

application/xml

JavaScript – document.form

Para referenciar o formulário, utiliza-se o Array `forms[]` ou o nome do objeto incluído no atributo “name” da tag `<form>` da HTML.

Ex.:

```
document.forms[0].title = "Mudei o título 1";  
document.forms["Formulario1"].title = "Mudei o título 2";  
document.forms.Formulario1.title = "Mudei o título 3";  
document.Formulario1.title = "Mudei o título 4";
```

DV0

É bom abordar o tratamento de formulários, mas a validação de formulários no lado do cliente é frequentemente feita com bibliotecas ou os recursos de validação integrados do navegador, em vez de apenas usar mensagens alert().

DENILCE DE ALMEIDA OLIVEIRA VEL; 2025-04-25T19:51:13.839

JavaScript – document.form.elements

O Array **elements** está subordinado a um formulário. É possível utilizar o nome do objeto, definido através do atributo `name` incluído na tag de cada elemento do formulário (exemplo: `<input>` `<textarea>`,`<button>`)

Propriedades

- ✓ `elements[].name` - Armazena o nome do elemento.
- ✓ `elements[].length` - Armazena o comprimento do elemento.

*** Ver exemplo UsandoDOM19Elements.html*

JavaScript – Navigator

Navigator - Apresenta informações do navegador.

Propriedades

- ✓ *navigator.appCodeName* - Nome interno do código do navegador, normalmente Mozilla.
- ✓ *navigator.appVersion* - Versão utilizada pelo navegador.
- ✓ *navigator.language* - Idioma. Ex.: "in" para inglês, "pt-br" para português do Brasil.
- ✓ *navigator.platform* - Plataforma do computador utilizado pelo navegador, como "Win32", etc.
- ✓ *navigator.cookieEnabled* - Permite cookies.
- ✓ *navigator.onLine* - Está online.

**** Ver exemplo:**

UsandoNavigator1.html

UsandoNavigator2.html

JavaScript – Navigator

✓ *navigator.userAgent* - Cabeçalho do usuário-agente, uma string que o navegador envia para o servidor Web quando solicita uma página da Web, e inclui detalhes do navegador como nome do navegador, a versão e informações sobre o sistema operacional.

DV0

Exemplo:

```
const userAgent = navigator.userAgent;
if (userAgent.indexOf('Firefox') !== -1) {
    console.log('Você está usando o navegador Firefox');
} else if (userAgent.indexOf('Chrome') !== -1) {
    console.log('Você está usando o navegador Google Chrome');
} else if (userAgent.indexOf('Safari') !== -1) {
    console.log('Você está usando o navegador Safari');
} else if (userAgent.indexOf('MSIE') !== -1 || userAgent.indexOf('Trident') !== -1) {
    console.log('Você está usando o navegador Internet Explorer');
} else {
    console.log('Não foi possível determinar o nome do navegador');
}
```

DV0

O objeto Navigator é útil, mas a detecção do navegador usando userAgent (como mostrado no exemplo) geralmente é desencorajada porque pode não ser confiável. A detecção de recursos é uma abordagem melhor. Por exemplo: Supondo que deseja-se usar a API fetch. Em vez de verificar se o navegador é "Chrome" ou "Firefox", verifica-se se o navegador suporta a função fetch:

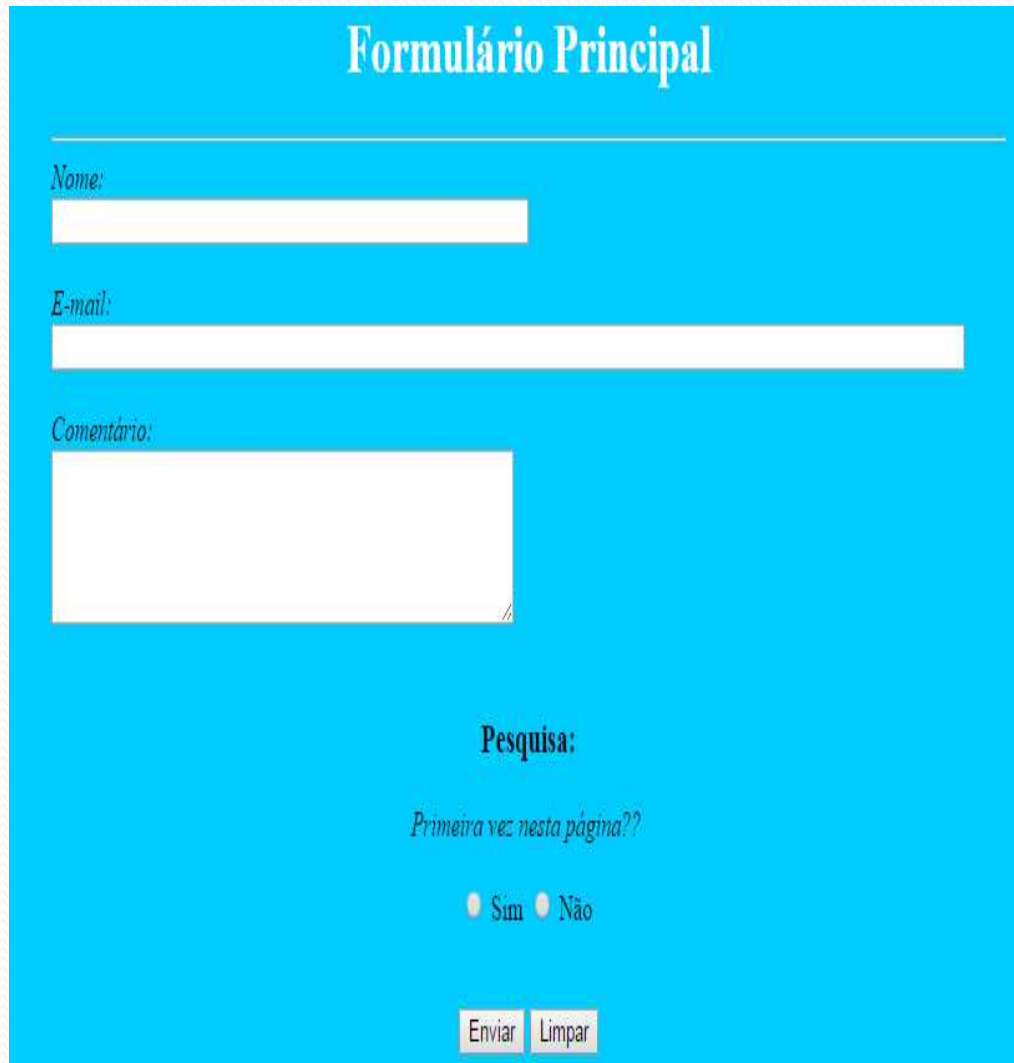
```
if ('fetch' in window) {  
  // O navegador suporta fetch  
  fetch('sua_url')  
    .then(response => { /* ... */ })  
    .then(data => { /* ... */ });  
} else {  
  // O navegador não suporta fetch  
  alert('Seu navegador não suporta fetch.');
```

}

DENILCE DE ALMEIDA OLIVEIRA VEL; 2025-04-25T20:05:29.646

Exercício 1

Criar o seguinte formulário.



The image shows a web form titled "Formulário Principal" on a blue background. It contains three input fields: "Nome:" (a single-line text box), "E-mail:" (a single-line text box), and "Comentário:" (a multi-line text area). Below these fields is a section labeled "Pesquisa:" with the text "Primeira vez nesta página??" and two radio buttons labeled "Sim" and "Não". At the bottom are two buttons: "Enviar" and "Limpar".

- ✓ Criar uma função validar no evento onsubmit do Form.
- ✓ Nome não pode ter menos que 10 caracteres.
- ✓ Email (type=email)
- ✓ Comentário deve ter no mínimo 20 caracteres.
- ✓ Pesquisa (obrigatório). Se não: Retornar : “Que bom que você voltou a visitar esta página!”, caso contrário: “Volte sempre à esta página!”. Utilize mesmo atributo name nos radios.
- ✓ Utilizar `document.nomeform.elements[]` na função – pelo menos em algum caso

Disponibilizar como Atividade15 no GITHUB.
→ Seuusuario/PWEB/Atividade15

Exercício 2

Criar uma página utilizando JavaScript que execute as seguintes funções:

- ✓ A página principal deve conter uma caixa de seleção com nomes de cursos (os cursos da Fatec Sorocaba).
- ✓ Quando o usuário escolher um curso, deverá aparecer uma caixa confirmando se a janela contendo o curso deve realmente ser aberta.
- ✓ Caso o usuário confirme (clicando em Ok), o curso escolhido deverá ser carregado em uma nova janela (coloque algumas informações sobre ele) com 600 × 300 pixels.
- ✓ Use o evento onchange do tag <select> para carregar o curso escolhido.

Disponibilizar como Atividade16 no GITHUB.

→ Seuusuario/PWEB/Atividade16

Atividade Extra – Cálculo de Média

Criar uma página com um formulário, que deve receber os dados de 10 alunos (um de cada vez): Nome Completo, RA, Nota 1, Nota 2, Nota 3. Valide os dados: Nome Completo não pode ser vazio, deve ter um nome e pelo menos 1 sobrenome, RA deve ter 5 dígitos (números), e as notas devem estar entre 0 e 10. Calcule a média do aluno. Na parte inferior da tela, ir mostrando, o nome do aluno, ra e a média, a cada entrada. Quando terminar de ler todos os dados de todos os alunos, mostrar a média geral dos mesmos.

Disponibilizar como AtividadeExtra no GITHUB.

→ Seuusuario/PWEB/AtividadeExtra

Características importantes do JS

Closure: *uma função pode acessar o escopo de sua função pai, mesmo após a função pai ter terminado de executar.*

```
function createCounter(initialValue) {  
  let count = initialValue;  
  return function() {// cria fechamento que mantém count  
    return count++; //possui capacidade manter estado interno - valor do contador  
  };  
}  
  
const counter1 = createCounter(0);  
alert(counter1()); // 0 - a primeira vez ainda não incrementado  
alert(counter1()); // 1  
alert(counter1()); // 2  
const counter2 = createCounter(10);  
alert(counter2()); // 10  
alert(counter2()); // 11
```


Características importantes do JS

Promises

São objetos que representam o resultado eventual de uma operação assíncrona. Elas podem estar em três estados: pendente (estado inicial), resolvida (sucesso) ou rejeitada (erro).

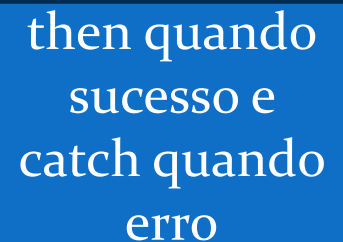
Permite que o código continue sua execução sem bloquear, enquanto espera o resultado da Promise. Vantagens: **melhora de performance, código mais moderno, legível e fácil de manter, especialmente para operações assíncronas complexas.**

Exemplo: uma Promise proporciona uma maneira de lidar com operações que levam tempo para serem concluídas, como solicitações de rede ou leitura de arquivos.

Características importantes do JS

Exemplo com Promises – lendo arquivo

```
const fs = require('fs').promises;  
fs.readFile('arquivo.txt', 'utf-8')  
  .then(contenido => console.log(contenido))  
  .catch(erro => console.error(erro));
```



then quando
sucesso e
catch quando
erro

Exemplo SEM Promises – lendo arquivo

```
const fs = require('fs');  
fs.readFile('arquivo.txt', 'utf-8', (erro, contenido) => {  
  if (erro) {  
    console.error(erro);  
    return;  
  } console.log(contenido);  
});
```


Características importantes do JS

Exemplo com Promises – chamadas de API

Promise está implicitamente presente através da função `fetch()` que é uma API JavaScript moderna para fazer requisições de rede (como buscar dados de um servidor).

Essa função sempre retorna uma Promise. Essa Promise representa a conclusão (ou falha) da requisição.

```
fetch('<https://api.exemplo.com/dados>')  
.then(response => response.json())  
.then(dados => console.log(dados))  
.catch(erro=>console.error(erro));
```


Características importantes do JS

Exemplo Sem Promises – chamadas de API

```
function buscarDados(callback) {  
  const xhr = new XMLHttpRequest();  
  
  xhr.open('GET', 'https://api.exemplo.com/dados');  
  
  xhr.onload = function() {  
    if (xhr.status >= 200 && xhr.status < 300) {  
      try {  
        const dados = JSON.parse(xhr.responseText);  
        callback(null, dados); // Chama o callback com os  
dados
```


Características importantes do JS

Exemplo Sem Promises – chamadas de API

```
function buscarDados(callback) {  
  const xhr = new XMLHttpRequest();  
  
  xhr.open('GET', 'https://api.exemplo.com/dados');  
  
  xhr.onload = function() {  
    if (xhr.status >= 200 && xhr.status < 300) {  
      try {  
        const dados = JSON.parse(xhr.responseText);  
        callback(null, dados); // Chama o callback com os dados  
      } catch (erro) {  
        callback(erro); // Chama o callback com o erro de parsing  
      }  
    } else {  
      callback(new Error(`Erro na requisição: ${xhr.status}`)); // Chama o callback com o  
      erro da requisição  
    }  
  };  
}
```


Características importantes do JS

```
xhr.onerror = function() {  
  callback(new Error('Erro de rede')); // Chama o callback com o erro de rede  
};
```

```
xhr.send();  
}
```

```
buscarDados((erro, dados) => {  
  if (erro) {  
    console.error(erro);  
    return;  
  }  
  console.log(dados);  
});
```


Características importantes do JS

Async/Await

São extensões das Promises, introduzidas para simplificar a escrita de código assíncrono. Uma função marcada como `async` retorna uma `Promise`, e a palavra-chave `await` é usada para esperar de forma síncrona o resultado de uma `Promise`, tornando o código mais legível e fácil de manter. `Await` pausa a execução da função `async` até que a `Promise` que está esperando seja resolvida.

- Se a `Promise` for cumprida, `await` retorna o valor resolvido.
- Se a `Promise` for rejeitada, `await` lança um erro, que pode ser capturado usando um bloco `try...catch`.
- Melhora significativamente a legibilidade e a manutenção do código assíncrono.

Características importantes do JS

Exemplo com Async/Await – chamada de API

```
async function buscarDados() {  
  try {  
    const response = await  
fetch('<https://api.exemplo.com/dados>');  
    const dados = await response.json();  
    console.log(dados);  
  } catch (erro) {  
    console.error(erro);  
  }  
}
```


Algumas Diferenças entre JavaScript e TypeScript

<https://www.youtube.com/watch?v=W1RhCMMKrqc>

https://www.alura.com.br/artigos/javascript-ou-typescript?gclid=CjwKCAjwndCKBhAkEiwAgSDKQdPykCacmyOdYHfA4pZu1PskwQWOX7ACYtqghQx6QDN6yM17Mr4NRoCDqYQAvD_BwE

Referências

CODEACADEMY. Cursos Gratuitos. <https://www.codecademy.com/pt> Acesso em: Jan. 2025.

CROCKFORD, Douglas. <http://crockford.com/javascript/> Acesso: Mai.2021.

CSS. <http://del.icio.us/carlosbazilio/{css+html}> Acesso em: Jan.2015.

DOM. Disponível em: http://www.w3schools.com/jsref/dom_obj_document.asp Acesso em: Jan.2025

HERANÇA. Disponível em: <https://www.devmedia.com.br/classes-no-javascript/23866> Acesso em: Jan.2025

JS. Livro de JavaScript. <https://github.com/getify/You-Dont-Know-JS> Acesso em: Jan.2021.

JS2. https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/JavaScript_Vis%C3%A3o_Geral Acesso em: Jan.2025

NODELIST. NodeList. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/API/NodeList> Acesso em: Abr.2025.

MDN. Resources for Developers, by Developers Disponível em: <https://developer.mozilla.org/pt-BR/> Acesso em:Abr.22025.

JSOBJETOS. https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Trabalhando_com_Objeto Acesso em: Jan.2025.

MOZILLA. JavaScript.. <https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript> Acesso em: Set.2024.

OPERADORES. https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Expressions_and_Operators#Assignment_operators Acesso em: Jan.2025.

PEREIRA. Fábio M. Pereira. JavaScript Básico. DESENVOLVIMENTO DE SISTEMAS WEB – 2014.1 UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA CURSO DE CIÊNCIA DA COMPUTAÇÃO. 2014.

PRECEDENCIA. [https://msdn.microsoft.com/pt-br/library/z3ks45k7\(v=vs.94\).aspx](https://msdn.microsoft.com/pt-br/library/z3ks45k7(v=vs.94).aspx) Acesso em: Jan.2025.

Referências

SANTOS. Elisabete da Silva. Apostila JavaScript - Faculdade de Tecnologia de São de Paulo.

SILVA. Maurício Samy . JavaScript Guia do Programador. Editora Novatec.

W3SCHOOLS. Disponível em: <http://www.w3schools.com/> Acesso em: Jan.2025.

W3. Disponível em: <http://www.w3.org/> Acesso em: Jan.2025..