

Informática
CURSO TÉCNICO

Programação WEB

Aula 01: Introdução ao JS

O que é o Javascript?

- 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
- Pode ser executada tanto do lado do navegador como do servidor (Node.js)
- Não é necessário compilar explicitamente o código JavaScript
- Não confundir com a linguagem de programação Java

Javascript e ECMAScript

- Ecma International - Organização que desenvolve padrões
- ECMAScript é uma linguagem padronizada, uma especificação
- ECMA262 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 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>

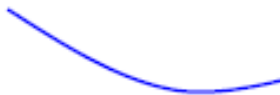
  </head>

  <body>

    ...

  </body>

</html>
```



Arquivo JS

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

Observações Gerais

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

Estruturas condicionais e de Repetição

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

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

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

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

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

Declaração de variáveis

`var nomeDaVariável= valorInicial`

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

`let nomeDaVariável= valorInicial`

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

`const nomeDaConstante= valor`

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

Exemplo de Variáveis

```
<script>
  const pi = 3.14
  var soma = 0; // soma é uma variável global
  for (let i = 1; i <= 10; i++) {
    soma += i
  }
  if (soma > 50) {
    let k = soma + pi; // k só pode ser acessada aqui
    var m = k + 1
    console.log(k)
  }
  console.log(m)
  console.log(k)
</script>
```

Operadores aritméticos, relacionais e lógicos

Operadores Aritméticos e Atribuição

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

Operadores Relacionais e Lógicos

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

Operador de adição e concatenação

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

Diferença dos operadores == e ===

- Operador ==
 - Compara apenas valores
- Operandos de tipos diferentes são convertidos e valores comparados
- Operador ===
 - Compara o valor e o tipo dos operandos
 - Operandos de tipos diferentes sempre resulta em falso

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

Objetos *window*, *navigator*, *document*

window

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

navigator(ou `window.navigator`)

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

document(ou `window.document`)

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

Métodos para E/S

`window.alert`

exibe uma caixa de diálogo para mensagens (botão Ok)

`window.confirm`

exibe uma caixa de diálogo para confirmação (Ok/Cancelar)

`window.prompt`

exibe uma caixa de diálogo para entrada de texto

`document.write`

adiciona conteúdo no documento HTML

`console.log`

registra conteúdo de *log* no **console** do navegador

`console.warn`

registra mensagem de *warning* no **console** do navegador

`console.error`

registra mensagem de *erro* no **console** do navegador

Métodos para E/S

`window.alert`

exibe uma caixa de diálogo para mensagens (botão Ok)

`window.confirm`

exibe uma caixa de diálogo para confirmação (Ok/Cancelar)

`window.prompt`

exibe uma caixa de diálogo para entrada de texto

`document.write`

adiciona conteúdo no documento HTML

`console.log`

registra conteúdo de *log* no **console** do navegador

`console.warn`

registra mensagem de *warning* no **console** do navegador

`console.error`

registra mensagem de *erro* no **console** do navegador

Declaração de funções

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

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

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

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

Arrays em javascript

- Não são tipos de dados primitivos
- São tratados como objetos, com propriedades e métodos
- Podem armazenar valores de tipos diferentes
- Os elementos são acessados por índice numérico (e não por chaves)
 - Não são associados

Arrays em javascript

- Elementos colocados entre colchetes, separados por vírgula

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

- Elementos de diferentes tipos

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

- Pode ser iniciado com vazio

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

Arrays em Javascript – Outros métodos

```
let vogais= ['E', 'I', 'O'];  
  
vogais.push('U')           // adiciona um item no final do vetor  
  
vogais.pop()               // remove e retorna o último item do vetor  
  
vogais.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
```

Percorrendo array com Estrutura **for**

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

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

Percorrendo array com Estrutura **forEach**

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

Percorrendo array com método **forEach** e função **anônima**

Função anônima é uma função sem nome. No caso acima, a função é um parâmetro do método **forEach**. Para cada elemento do array, será executada a função

Percorrendo array com Estrutura **forEach**

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

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

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

Arrow function é uma forma mais sucinta de se fazer uma função. Existem várias formas de se fazer, dependendo da quantidade de parâmetros e comandos da função.

No exemplo acima, **elemento** é o parâmetro da função.

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 String

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

```
<script>
  let a = 1
  let b = 2
  let c = 3
  const delta = b*b - 4*a*c
  console.log(`O discriminante da equação com
    coeficientes ${a}, ${b} e ${c} é ${delta}`)
</script>
```


Objetos simples (plain object, POJO)

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

Objetos simples (plain object, POJO)

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

Tratamento de eventos

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

Tratamento de eventos

Propriedades de Tratamentos de Eventos

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

```
//evento load ocorre quando a página inteira é carregada
window.load = function(){
    console.log('Pagina carregada!')
}
```

Método *addEventListener*

Adiciona uma função a ser executada na ocorrência de um evento:

```
// 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
window.addEventListener('load', function(){
    // ...
})
```

Tratamento de eventos - exemplo

```
<body>

  <script>

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

    window.onload = mostraMsg;

  </script>

</body>
```

Utilizando a propriedade de evento `onload`

```
<body>

  <script>

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

    window.addEventListener('load', mostraMsg);

  </script>

</body>
```

Utilizando o método `addEventListener`

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", funcaoCallback)
```

Evento *DOMContentLoaded*

```
<body>

  <script>

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

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

    });

  </script>

</body>
```

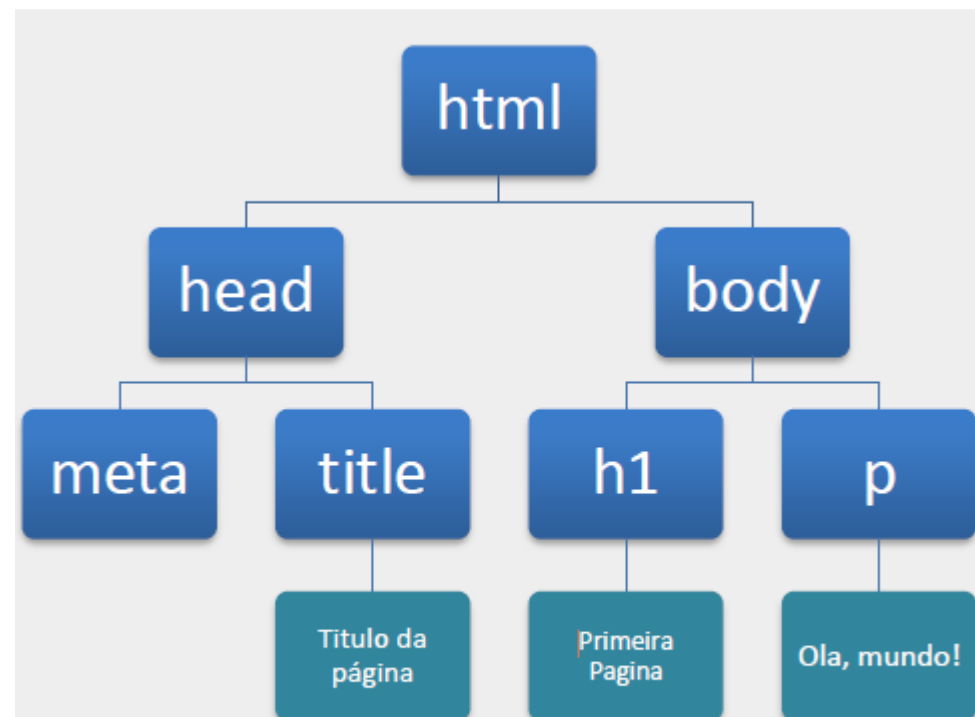
Manipulação da Árvore DOM

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

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

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

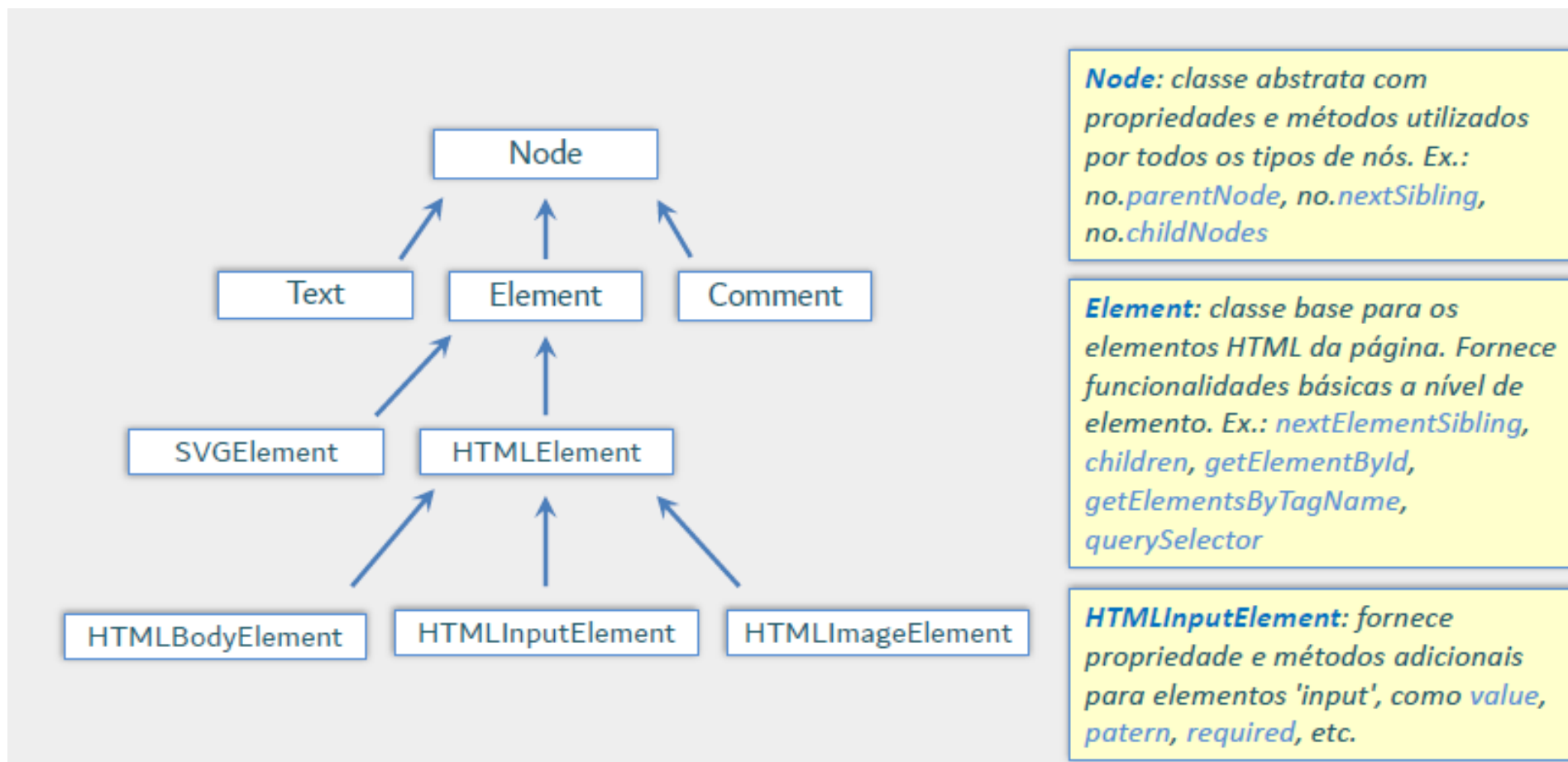
</html>
```



Abstração da Árvore DOM correspondente

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

Tipos de objetos na Árvore DOM



Hierarquia de nós na estrutura DOM

Nó **Root**: nó representando o elemento raiz `<html>`

Nó **Filho**: nó representando um elemento diretamente dentro de outro

Nó **Pai**: nó representando o elemento que contém o nó filho

Nós **Irmãos**: nós representando elementos filhos do mesmo pai

Manipulação Árvore DOM

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

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

Busca na 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

Busca na árvore DOM - `document.querySelector`

Retorna o nó correspondente ao primeiro elemento h1 na página

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

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

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

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

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

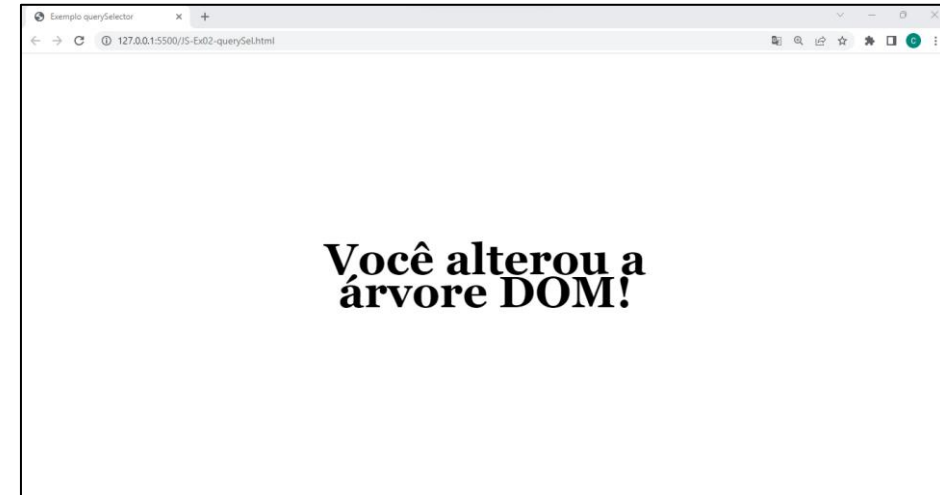
Exemplo

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

<script>

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

</script>
```



Busca na árvore DOM - `document.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

Continua...