



# Programação Client-Side – JavaScript

Prof. Mizael Carlos

# O que é JavaScript?



- É uma linguagem de programação
  - Não confundir com a linguagem de programação Java!
  - Comumente referenciada como JS
- Utilizada para prover **interatividade** e **dinamismo** a websites
- Comumente executada no lado cliente, pelo navegador Web
  - 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

# O que é JavaScript?

3

- Permite programar o comportamento da página Web na ocorrência de eventos
  - Executar ações para inicializar a página Web assim que ela é carregada
  - Executar operações em resposta a ações do usuário, como “clicar em um botão” ou “selecionar uma opção”
  - Validar o conteúdo de campos de formulários à medida que o usuário os preenche
- Permite alterar o documento HTML por meio da manipulação da árvore DOM
  - Alterar o conteúdo e o layout da página Web em tempo de exibição
  - Adicionar novos elementos, ocultar/exibir elementos, modificar atributos de elementos, ...

# JavaScript e ECMAScript

4

- Ecma International – organização que desenvolve padrões
- ECMAScript é uma linguagem padronizada, uma especificação
  - ECMA-262 é o nome do padrão propriamente dito
  - <https://www.ecma-international.org/publications-and-standards/standards/ecma-262/>
- JavaScript é uma implementação da linguagem ECMAScript
- JavaScript originalmente desenvolvida por Brendan Eich da Netscape (um dos precursores dos navegadores web) em 1995

# Execução do código JavaScript no navegador

5

- Cada navegador tem seu próprio motor JavaScript
  - Google Chrome: V8
  - Microsoft Edge: V8 (antigamente usava o Chakra)
  - Firefox: SpiderMonkey
  - Safari: JavaScriptCore
- Quando o motor JavaScript escaneia o arquivo com o código (script), um ambiente chamado de **contexto de execução** é criado
  - Nesse momento, memória é alocada para as variáveis e funções
  - Dois tipos de contexto de execução: global e de função

# Fases do contexto de execução JavaScript

6

Fases aplicáveis tanto no contexto global quanto de função.

## 1. Fase de **criação**

- O motor JS cria o contexto de execução e configura o ambiente do script. Determina os valores de variáveis e funções e configura a cadeia de escopo para o contexto de execução
- O código JS inserido com `<script>` é executado durante a fase de carregamento do documento HTML
- O motor JS associa o documento HTML no objeto `document` e a janela (do navegador) é associada ao objeto `window`

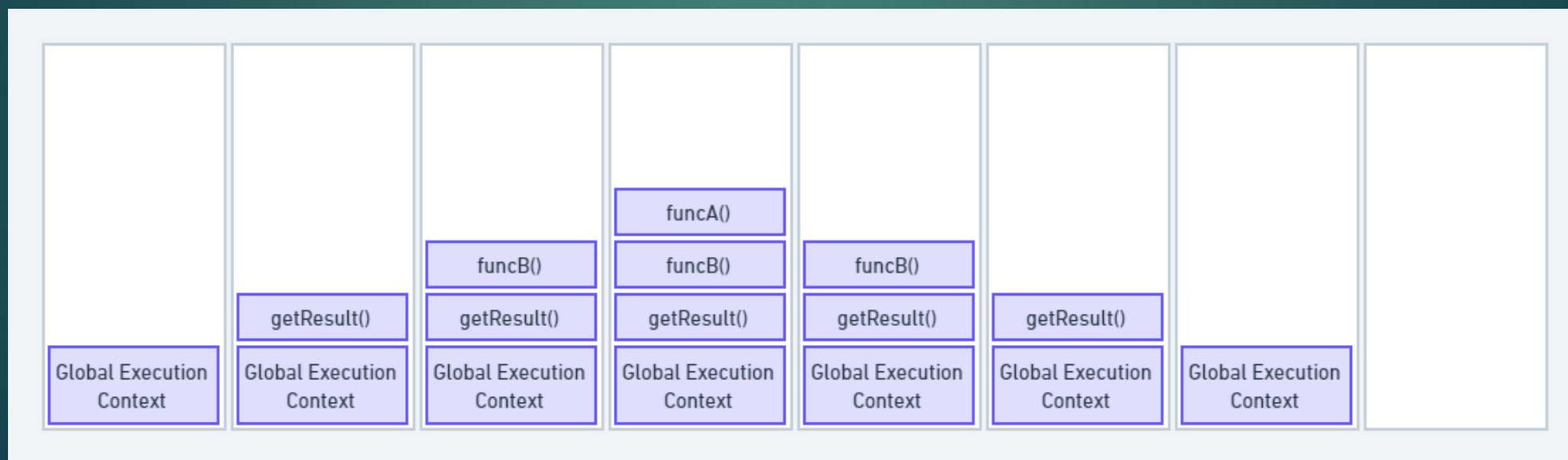
## 2. Fase de **execução**

- O motor JS executa o código no contexto de execução, linha por linha, de cima para baixo. Ele processa quaisquer instruções ou expressões no script e avalia quaisquer chamadas de função

# Pilha de contexto de execução

Cada navegador tem seu tamanho fixo de pilha. Se o número de contexto excede o tamanho, então ocorre um erro de *stack overflow*.

- Para acompanhar todos os contextos, inclusive globais e funcionais, o motor JavaScript usa uma pilha de chamadas
- Usa o princípio LIFO (Last-In-First-Out), ou seja, a última execução a ser empilhada é a primeira a ser desempilhada





# Formas de execução

8

## 1. Execução **síncrona**

- O motor JS executa sequencialmente e já retorna o resultado
- Uma instrução tem que esperar que a instrução anterior seja executada

## 2. Execução **assíncrona**

- Faz uma chamada para executar um trecho de código, mas não fica esperando o resultado (não bloqueia a execução do restante do código)
- Após toda a execução do código, o trecho é colocado na pilha de chamadas e, finalmente, é executado
- Código executado na ocorrência de eventos como click de botão, rolagem da página, dado de rede disponível etc.
- As funções a serem executadas geralmente são registradas na fase de criação do contexto



# JavaScript no navegador - *Thread*

9

- O navegador executa o código JS em modo de ***thread* única** (*single-threaded*)
  - Durante a execução do código JS, o navegador não responde à interface do usuário (exceto em requisições Ajax)
  - Pode causar uma experiência ruim de navegação ao executar operações que demoram para finalizar
- É possível executar código JS em *background* (outra *thread*) usando *web workers*, mas com acesso limitado ao contexto da *thread* principal

# Incorporar código JS na página HTML

10

Duas maneiras:

## 1. Embutido no HTML

- Mais simples e direto
- Código acessível somente dentro de uma página específica
- Dificulta a manutenção de sistemas maiores

## 2. Em arquivo separado

- Melhor separação de responsabilidades entre conteúdo (HTML) e comportamento (código JS)
- HTML conciso - código mais fácil de ler e manter
- Possibilita reutilizar o código JS em vários arquivos HTML
- Arquivos JavaScript podem ser mantidos em cache pelo navegador

# Código JavaScript embutido no HTML

11

```
<head>
```

```
  <title>Aqui vai o título da página</title>
```

```
  <script type="text/javascript">
```

```
    //meu código Javascript
```

```
    ...
```

```
  </script>
```

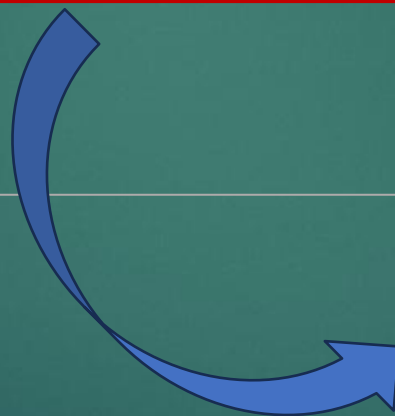
```
</head>
```

# Código JavaScript em arquivo separado

12

*pagina.html*

```
...  
<head>  
  <title>Aqui vai o título da página</title>  
  <script src="meujavascript.js"></script>  
</head>  
...
```



*meujavascript.js*

```
//meu código Javascript  
  
...
```

# Observações gerais sobre a linguagem

13

- 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
  - linguagem dinâmica com tipos dinâmicos
  - linguagem de tipagem fraca
- Comentários de linha: `// comentário`
- Comentários de bloco: `/* comentário */`

# Algumas propriedades e métodos gerais

14

**window:** Representa a aba do navegador que contém a página e possibilita obter informações ou realizar ações a respeito da janela

- `window.innerHeight` : retorna a altura do conteúdo da página
- `window.innerWidth` : retorna a largura do conteúdo da página
- `window.alert()` : mostra uma caixa de alerta com uma mensagem e um botão OK
- `window.confirm()` : mostra uma caixa de diálogo para confirmação (botões OK e cancelar)
- `window.close()` : fecha a janela

**document (ou window.document)** : Representa o documento HTML carregado na aba do navegador e possibilita a manipulação da árvore DOM

- `document.URL` : retorna a URL completa do documento HTML
- `document.title` : retorna o título da página HTML
- `document.write()` : escreve um texto no documento HTML

# Algumas propriedades e métodos gerais

15

**navigator** (ou **window.navigator**): Representa o navegador de Internet em uso (browser, user-agent) e fornece informações como idioma do navegador, geolocalização, memória, etc.

- `navigator.language`: retorna o idioma do navegador ("en", "en-US", "pt", "pt-BR", ...)
- `navigator.geolocation`: retorna um objeto de geolocalização para a localização do usuário
- `navigator.platform`: retorna a plataforma do navegador ("Linux i686", "Mac68K", "Win32", "WebTV OS", ...)

**console** (ou **window.console**): Representa o console de debug do navegador

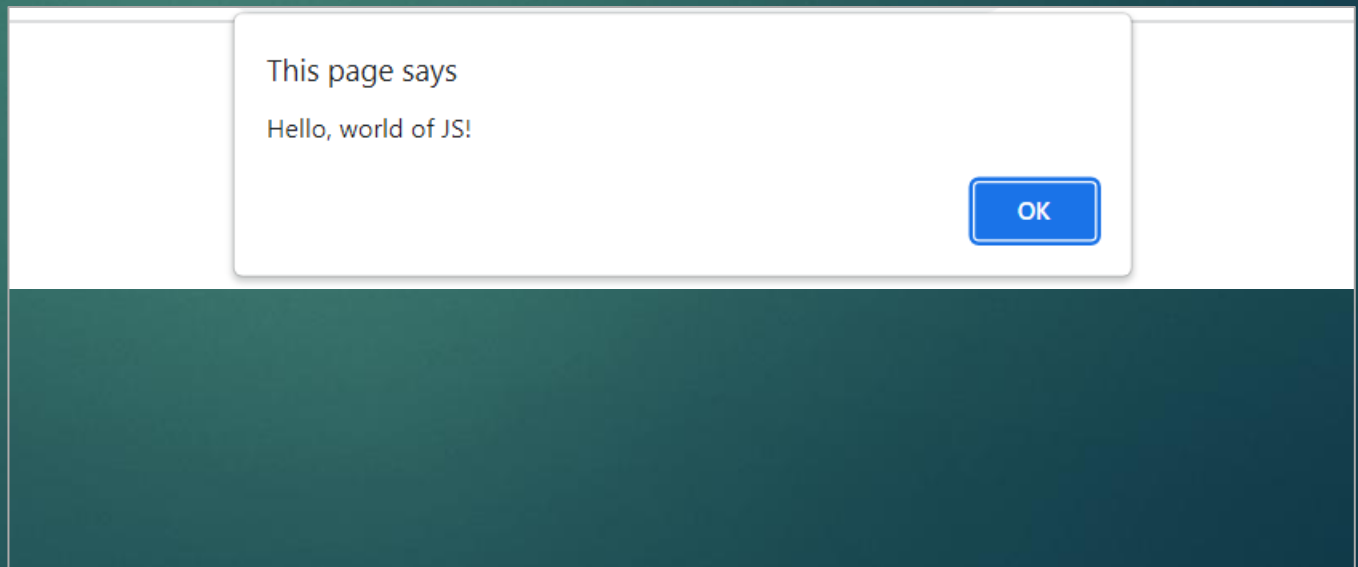
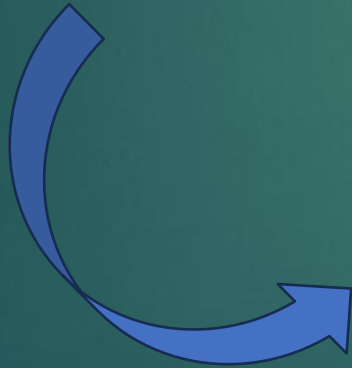
- `console.log()`: registra conteúdo de log no console do navegador



# Exemplo 1

16

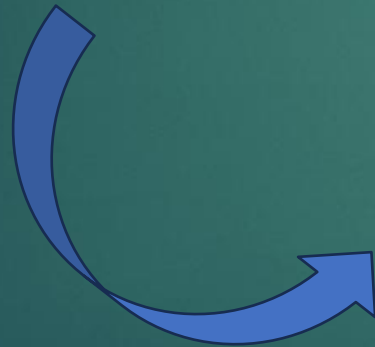
```
<script type="text/javascript">  
    alert("Hello, world of JS!");  
</script>
```



# Exemplo 2

17

```
<script type="text/javascript">  
  document.write("A altura do conteúdo da janela é: " + window.innerHeight + "px. <br/>");  
  document.write("A largura do conteúdo da janela é: " + window.innerWidth + "px. <br/>");  
</script>
```



A altura do conteúdo da janela é: 739px.  
A largura do conteúdo da janela é: 1536px.

# Exemplo 3

18

```
<script type="text/javascript">  
    alert(document.URL);  
    alert(document.title);  
  
    document.write("A URL desta página é: " + document.URL);  
</script>
```

- Os tipos das variáveis são definidos automaticamente
  - linguagem dinâmica com tipos dinâmicos (podem ser alterados em tempo de execução)
  - linguagem de tipagem fraca, ou seja, não ocorre verificação de tipos nas operações efetuadas
- O nome deve começar por uma letra ou pelo caractere “\_”
- Como já mencionado, é *case-sensitive* (teste, Teste e TESTE são diferentes)
- Podem ser definidas tanto no escopo global quanto de função (local)

# Variáveis

20

- Não é necessário declarar o tipo de uma variável
- A variável irá “alterar” o seu tipo de dado conforme os valores forem atribuídos

```
var x; // x é indefinido  
x = 5; // x é um número  
x = "John"; // x é uma string  
x = true; // x é um valor lógico  
x = null; // x é indefinido
```

# Declaração de variáveis

21

**var** nomeDaVariavel = 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, mas não pode ser atualizada
- Deve ser inicializada no momento da declaração

Blocos de código  
são delimitados  
por chaves: { }

# Operadores aritméticos e atribuição

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

$x++$  é o mesmo que  $x = x + 1$

$x--$  é o mesmo que  $x = x - 1$

$x += 10$  é o mesmo que  $x = x + 10$

$x -= 10$  é o mesmo que  $x = x - 10$



# Operadores de relacionais e lógicos

23

Operação	Operador
Comparação por igualdade	==
Diferente	!=
Estritamente igual (valor e tipo)	===
Estritamente diferente	!==
Menor que	<
Menor ou igual a	<=
Maior que	>
Maior ou igual a	>=

Operação	Operador
E	&&
OU	
NOT	!
Coalescência nula	??

Retorna o operando do lado direito quando o seu operador do lado esquerdo é null ou undefined. Caso contrário, retorna o operando do lado esquerdo.  
Ex.: `const foo = null ?? 'default string';`

# Tabela verdade operado lógico &

Condição 1	Condição 2	Resultado
Verdade	Verdade	Verdade
Verdade	Falso	Falso
Falso	Verdade	Falso
Falso	Falso	Falso

# Tabela verdade operado lógico ou

Condição 1	Condição 2	Resultado
Verdade	Verdade	Verdade
Verdade	Falso	Verdade
Falso	Verdade	Verdade
Falso	Falso	Falso

# Operador de adição e concatenação

26

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

# Operadores == e ===

27

- Operador ==
  - Compara apenas valores (conteúdo)
  - Operandos de tipos diferentes são convertidos e valores comparados
- Operador ===
  - Compara o valor e o tipo dos operandos
  - Operandos de tipos diferentes sempre resulta em falso
- Exemplos:
  - `1 == true; // retorna true, pois true é convertido para 1`
  - `1 === true; // retorna false;`
  - `10 == "10" // retorna true depois de converter 10 para string`
  - `10 === "10" // retorna false;`

# Controle condicional (if-else)

28

- Uma declaração condicional é um conjunto de comandos que são executados caso uma condição especificada seja verdadeira

```
if (condicao) {  
    //instrução 1  
    //instrução 2  
} else {  
    //instrução 3  
    //instrução 4  
}
```

```
if (condicao) {  
    //instrução 1  
    //instrução 2  
} else if (condição_2) {  
    //instrução 3  
    //instrução 4  
} else {  
    //instrução 5  
    //instrução 6  
}
```

# Controle condicional (switch-case)

- Uma declaração switch permite que um programa avalie uma expressão e tente associar o valor da expressão ao rótulo de um case. Se uma correspondência for encontrada, o programa executa a declaração associada.

```
switch (expressao) {  
    case rotulo_1:  
        //instrução 1  
        //instrução 2  
        break;  
    case rotulo_2:  
        //instrução 3  
        //instrução 4  
        //break;  
    ...  
    default:  
        //instrução y  
        //instrução z  
        //break;  
}
```



# Controle de repetição

30

- Laços (loops) oferecem um jeito fácil e rápido de executar uma ação repetidas vezes
- Várias formas de implementação

```
for (var i = 0; i < 5; i++) {  
    ....  
}
```

```
while (i < 5) {  
    i++;  
    ...  
}
```


```
do {  
    i += 1;  
    ....  
} while (i < 5);
```

# Exemplo 4 - repetição

31

```
<script type="text/javascript">  
    for (var i = 0; i < 5; i++) {  
        alert("Hello, world " + i + "!");  
    }  
</script>
```

- Em JavaScript, um array é uma estrutura de dados que permite armazenar uma coleção ordenada de valores, acessíveis por um índice numérico.
- É como uma variável que pode conter múltiplos elementos de diferentes tipos (números, strings, objetos, etc.).
- Os elementos são acessados por índice numérico
  - O primeiro elemento do array possui índice 0

- 
- São tratados como objetos, com propriedades e métodos
    - Por exemplo, o número de elementos pode ser resgatado por meio da propriedade `length`

- Elementos colocados entre colchetes, separados por vírgula
  - `let pares = [2, 4, 6, 8];`
  - `let primeiroPar = pares[0]; // 1º elemento`
  - `let nroElementos = pares.length; // tamanho do vetor`
- Elementos de diferentes tipos
  - `let vetorMisto = [2, 'A', true];`
- Pode ser iniciado com vazio
  - `let pares = [];`
  - `let pares = array();`

# Exemplo 5 - arrays

35

```
var vetorDeNumeros = [4, 1, 5, 2, 6];  
var soma = 0;  
var n = vetorDeNumeros.length;  
  
for (var i = 0; i < n; i++)  
    soma = soma + vetorDeNumeros[i];  
  
alert("A soma dos elementos do vetor é: " + soma);
```

# Arrays – alguns métodos

36

- ▶ `let vogais = ['E', 'I', 'O'];`
  - `vogais.push('U')` : adiciona um item no **final** do vetor
  - `vogais.pop()` : remove e retorna o **último** item do vetor
  - `vogais.shift()` : remove e retorna o **primeiro** item do vetor
  - `vogais.unshift('A')` : adiciona um item no **início** do vetor
  - `vogais.indexOf('E')` : retorna a posição da 1ª ocorrência de um item (ou -1)



# Percorrendo arrays

37

- É possível percorrer utilizando as estruturas `for` e `forEach`

```
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);  
};
```

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

# Objeto simples (plain object)

38

- Plain Old JavaScript 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
```

# Objeto **Math**

39

Método	Descrição
Math.sqrt(x)	Retorna a raiz quadrada de x.
Math.pow(x,y)	Retorna o valor de x elevado a y.
Math.PI	Retorna o valor da constante matemática PI.
Math.sin(x)	Calcula o seno do angulo x, dado em radianos.
Math.cos(x)	Calcula o cosseno do angulo x, dado em radianos.
Math.tan(x)	Calcula a tangente do angulo x, dado em radianos.
Math.round(x)	Retorna o valor de x arredondado para o inteiro mais próximo.
Math.random()	Retorna um número fracionário aleatório entre 0 e 1.

# Strings

40

- Definida com aspas simples ou duplas
  - `let msg = "JavaScript";`
- Acessando um caracter
  - `let primeiraLetra = msg[0];`
  - `let primeiraLetra = msg.charAt(0);`
- Contra-barra para caracteres especiais
  - `let msg = 'It\'s ok';`
- Strings com aspas duplas podem conter aspas simples e vice-versa
  - `let msg = "It's ok";`
- Várias propriedades e métodos
  - `length`, `indexOf`, `substr`, `split`, etc.

# Template Literal (ou Template String)

41

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

```
let a = 1;
let b = 2;
let c = 3;
const delta = b*b - 4*a*c;
console.log(`o discriminante da equacao com coeficientes ${a}, ${b} e ${c} é ${delta}`);
```

# Funções

42

- São trechos de códigos comuns criados para serem reutilizados
- Exige a palavra reservada `function`
- Exemplos:
  - função para somar dois números
  - função para calcular o valor do frete de um produto
  - função para ordenar uma lista
  - função para salvar um dado em um banco
  - função para verificar a disponibilidade de um recurso

# Declaração de funções

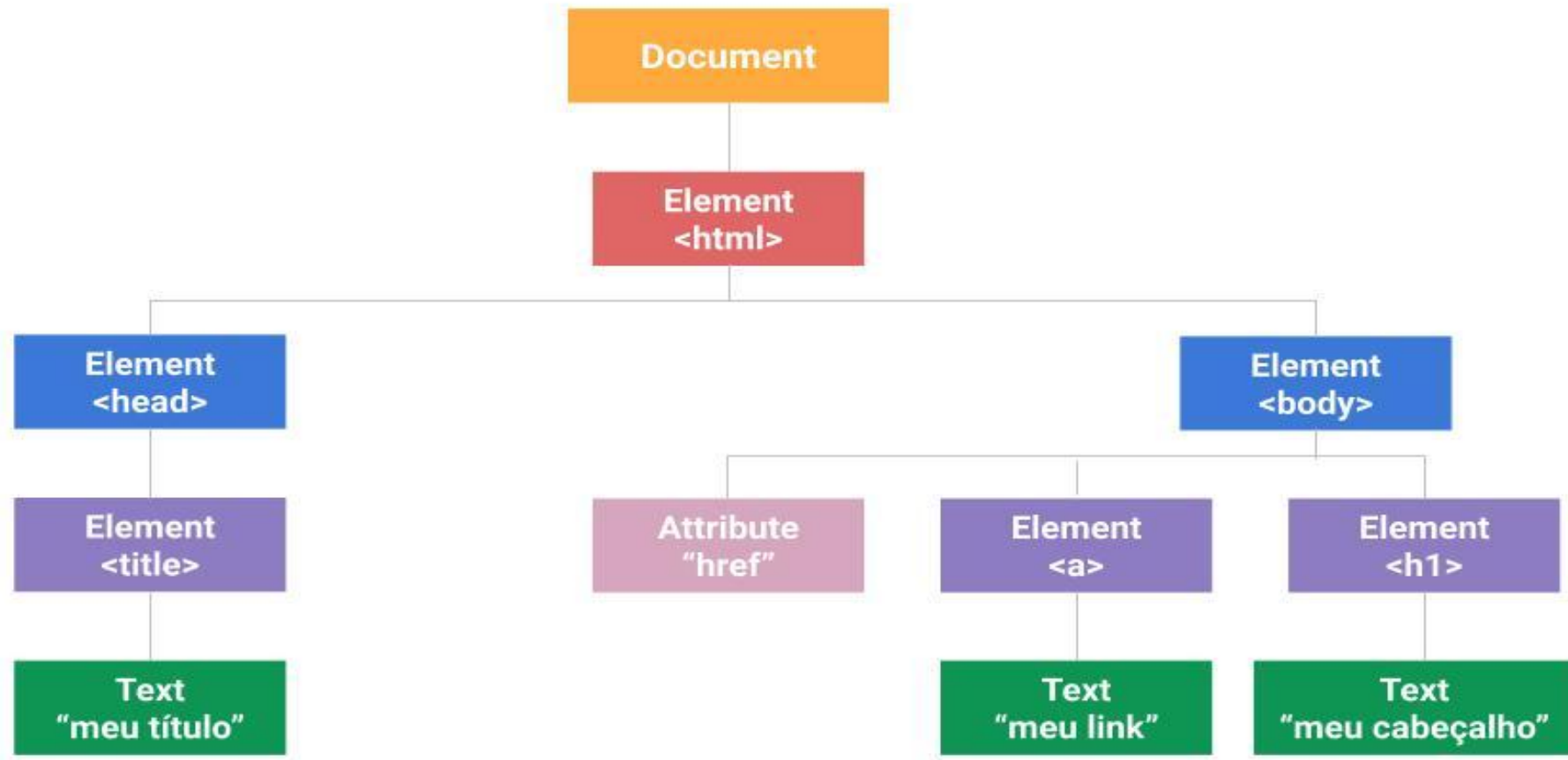
43

```
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





- 
- ▶ 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 JS.

# Hierarquia de nós na estrutura DOM

46

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

# Busca na árvore DOM

47

## `document.querySelector`

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

# Busca na árvore DOM

48

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");
```

# Função de callback

49

é uma função que é passada como argumento para outra função, e é executada em um ponto específico dentro dessa função, geralmente após uma operação assíncrona ou quando um evento ocorre

# Função de callback

50

JavaScript



```
function saudacao(nome, callback) {  
    var mensagem = "Olá, " + nome + "!";  
    callback(mensagem); // Chama a função callback passando a mensagem  
}  
  
function exibirMensagem(mensagem) {  
    console.log(mensagem);  
}  
  
// Passa a função exibirMensagem como callback  
saudacao("João", exibirMensagem); // Saída: Olá, João!
```

# Função de callback

51

- saudação é o nome da função que recebe um nome e uma função de callback.
- exibirmensagem é a função de call-back.
- Quando saudação é chamada com o nome “João” e exibirmensagem, a mensagem é construída e a função de call-back é executada exibindo a mensagem no console.

# Busca na árvore DOM

52

## `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

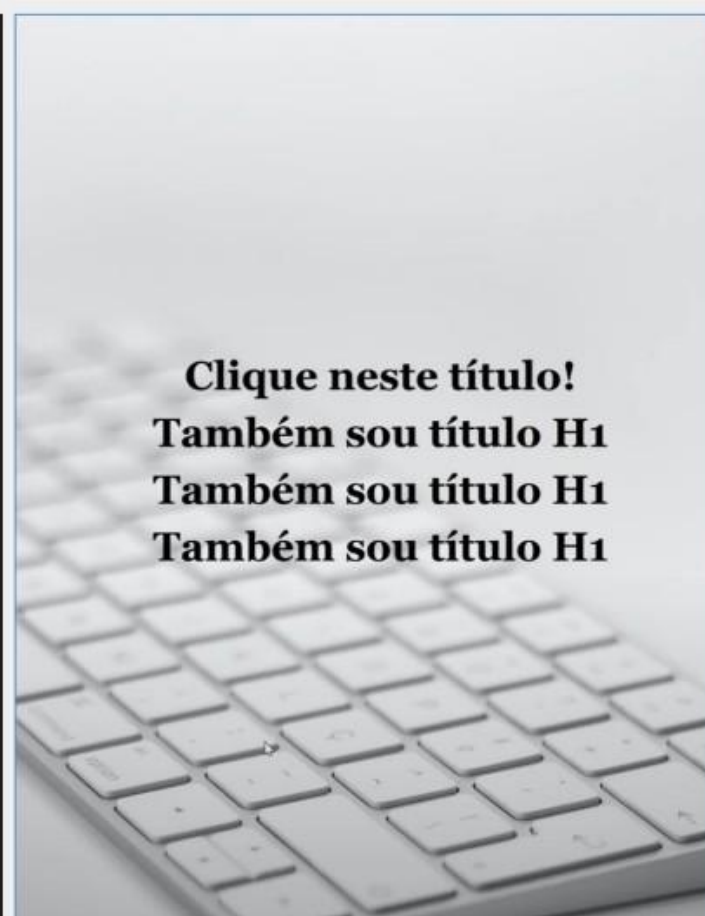
53

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

# Busca na árvore DOM - Exemplo

54

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



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

# Busca na árvore DOM - Exemplo

55

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

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

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

# Busca na árvore DOM - Exemplo

56

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

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

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

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

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

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



# Detalhes do evento

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

# Outras formas de busca na árvore DOM

58

- `document.getElementById`
  - busca um único elemento utilizando o seu **id**
- `document.getElementsByName`
  - busca os elementos pelo valor do **atributo name** do elemento
- `document.getElementsByTagName`
  - busca os elementos pelo **nome da tag** HTML, como `img`, `h1`, etc.
- `document.getElementsByClassName`
  - busca os elementos pelo **valor do atributo class**

# Acesso ao conteúdo dos elementos HTML

59

- Propriedade **textContent**
  - Se o conteúdo do elemento é textual, retorna esse texto
  - Se o elemento possui filhos, retorna a **concatenação** do textContent dos filhos
  - Uma alteração do valor removerá todos os nós filhos e **substituirá** pelo novo texto
- Propriedade **innerText**
  - Semelhante a textContent, porém não inclui conteúdos de elementos que “não podem ser lidos” pelo usuário, como conteúdo oculto com CSS, conteúdo de tags como `<script>`, `<style>` etc.
- Propriedade **innerHTML**
  - Retorna o conteúdo do elemento e de seus descendentes, incluindo as tags HTML
  - Quando alterada, o novo conteúdo é avaliado pelo navegador e pode resultar na criação de nós descendentes na estrutura DOM
  - OBS: possibilidade de ataques XSS e desempenho inferior a textContent.

# Método

60

## `element.insertAdjacentHTML()`

- Cria e insere nós na árvore DOM a partir de avaliação de string HTML
- Permite indicar a posição de inserção
- Sintaxe: `insertAdjacentHTML(posicao, textoHtml)`, onde `posicao` pode ser:
  - "beforebegin"
  - "afterbegin"
  - "beforeend"
  - "afterend"

```
<!-- beforebegin -->
<p>
  <!-- afterbegin -->
  Parágrafo de exemplo.
  <!-- beforeend -->
</p>
<!-- afterend -->
```



# Alterando estilos CSS de forma *inline*

61

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

CSS	JavaScript
<code>color</code>	<code>node.style.color</code>
<code>font-family</code>	<code>node.style.fontFamily</code>
<code>background-color</code>	<code>node.style.backgroundColor</code>

# Manipulando atributos

62

- 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>
```

# Arrow function =>

63

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

```
window.onload = function(){  
    alert("Hello!");  
    console.log("Hello!");  
};
```



```
window.onload = () => {  
    alert("Hello!");  
    console.log("Hello!");  
};
```

# Tratamento de eventos

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

# Tratamento de eventos

- Propriedades de tratamentos de eventos
  - Permite indicar uma função a ser executada na ocorrência de um evento

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

- Método `addEventListener`
  - Adiciona uma função a ser executada na ocorrência de um evento específico

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

# Tratamento de eventos - exemplos

*Usando a propriedade onload e função tradicional*

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

```
window.onload = mostraMsg;
```

```
window.onload = function(){  
    alert("Hello!");  
    console.log("Hello!");  
};
```

*Usando a propriedade onload e função anônima*

*Usando o método addEventListener e função tradicional*

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

```
window.addEventListener("load", mostraMsg);
```

```
window.addEventListener("load", function(){  
    alert("Hello!");  
    console.log("Hello!");  
});
```

*Usando o método addEventListener e função anônima*

# Tratamento de eventos

- Existem vários tipos de eventos que podem ser associados
  - Mouse: click, mouseover, mouseout, mousedown, mouseup, etc.
  - Teclado: keydown, keypress, keyup.
  - Formulário: focus, blur, change, submit.
  - Janela: load, unload, resize, scroll.



# Eventos mouse

- Click – Acionado no click do mouse.
- Mouseover - é um evento que ocorre quando o ponteiro do mouse é movido para dentro de um elemento HTML.
- Mouseout - O evento mouseout em JavaScript é acionado quando o cursor do mouse se move para fora de um elemento HTML.



# Eventos mouse

- Mousedown - é disparado quando um botão do mouse (ou de outro dispositivo apontador) é pressionado sobre um elemento HTML. Diferentemente do evento click, que ocorre após a pressão e liberação do botão, o mousedown é disparado no momento em que o botão é pressionado, ainda que o usuário não o libere.

# Eventos mouse

- Mouseup - O evento mouseup em JavaScript é disparado quando o botão do mouse é solto sobre um elemento. Ele ocorre após o evento mousedown, que é disparado quando o botão do mouse é pressionado. O evento mouseup é frequentemente usado para realizar ações quando o usuário completa uma interação de clique e solta o botão.

# Eventos teclado

- Keydown - é acionado quando uma tecla é pressionada no teclado, independentemente se produz um caractere ou não.
- Keypress - Disparado quando uma tecla que produz um caractere (como letras, números e pontuação) é pressionada. Este evento é útil para capturar a entrada de texto em formulários.

# Eventos teclado

- Keyup - Disparado quando uma tecla é liberada (soltada). Este evento é menos comumente usado, mas pode ser útil para detectar quando o usuário termina de digitar uma tecla ou quando deseja desativar uma função que foi acionada por uma tecla pressionada.

# Eventos formulário

- Focus - refere-se à funcionalidade que destaca um elemento HTML para indicar que ele está ativo e pronto para receber entrada do usuário, como texto em um campo de entrada ou clique em um botão.
- Blur - é um evento que é disparado quando um elemento perde o foco.

# Eventos formulário

- Change - é acionado quando o valor de um elemento HTML específico é alterado e o elemento perde o foco, ou seja, quando o usuário deixa de interagir com ele. Este evento é específico para elementos `<input>`, `<select>` e `<textarea>`.

# Eventos formulário

- Submit - refere-se ao processo de envio de dados de um formulário HTML para um servidor ou para realizar ações dentro do seu script. O evento "submit" é acionado quando o formulário é submetido, permitindo que você execute validações e outras ações antes do envio real. .



# Eventos janela

- Load - Este evento é disparado quando toda a página, incluindo imagens, CSS e scripts, está totalmente carregada.. unload, resize, scroll.



# Eventos janela

- beforeunload - é disparado quando a página está prestes a ser descarregada, geralmente quando o usuário está navegando para outra página ou fechando o navegador.

# Eventos janela

- Resize - "resize" refere-se ao evento que é disparado quando a área de visualização (document view) de um documento é redimensionada.

# Eventos janela

- Resize - "resize" refere-se ao evento que é disparado quando a área de visualização (document view) de um documento é redimensionada.

# Eventos janela

- Scroll - é acionado quando o usuário rola uma página ou elemento rolável .

# Manipulando atributos

81

- Alguns atributos são acessados de forma diferenciada

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

# Manipulando atributos

82

## `node.getAttribute`

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

# Manipulando atributos

83

## node.getAttribute

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

# Manipulando atributos

84

## node.getAttribute

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



# Manipulando atributos

85

## `node.setAttribute`

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

# Manipulando atributos

86

`node.setAttribute`

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

# Manipulação da árvore DOM

87

## `node.firstChild`

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

## `node.firstElementChild`

- retorna o primeiro nó filho do **tipo elemento**

## `node.lastChild`

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

## `node.lastElementChild`

- retorna o último nó filho do **tipo elemento**

# Manipulação da árvore DOM

88

`node.nextSibling`

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

`node.previousSibling`

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

`node.nextElementSibling`

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

`node.previousElementSibling`

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

# Manipulação da árvore DOM

89

## `node.hasChildNodes`

- retorna verdadeiro caso o nó tenha filhos

## `node.childNodes`

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

## `node.children`

- retorna lista contendo apenas nós filhos do tipo elemento

## `node.parentNode`

- retorna o nó pai do nó em questão

# Manipulação da árvore DOM

90

`node.appendChild(novoNo)`

- acrescenta um nó filho no final da lista de filhos

`node.removeChild(noFilhoASerRemovido)`

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

`node.remove()`

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

# Manipulação da árvore DOM

91

`document.createElement("elementoASerCriado")`

- cria um novo nó do tipo Element

`node.cloneNode(deep)`

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

# Outras propriedades do objeto `document`

92

## `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.



# Collections do Objeto document

## 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`

94

```
<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"];`

# Referências

95

- <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- <https://www.ecma-international.org/ecma-262/>