JavaScript (I e II): variáveis, tipos, operadores, funções, DOM, eventos, manipulação de elementos.

QXD0020 - Desenvolvimento de Software para Web

Universidade Federal do Ceará - *Campus* Quixadá

Prof. Francisco Victor da Silva Pinheiro
victorpinheiro@ufc.br

Agenda

- Introdução ao JavaScript
- Variáveis
- Escopo
- Como o Escopo Funciona
- Tipos de Dados
- Operadores
- Funções
- DOM (Document Object Model)
- Classes e Estilos Dinâmicos
- Eventos

Introdução ao JavaScript

- Criado em 1995 por Brendan Eich (Netscape).
- Linguagem interpretada, dinâmica, multi-paradigma (suporta procedural, OO e funcional).
- Base da Web moderna: manipulação do navegador, páginas dinâmicas, interatividade.
- Hoje vai além do front-end (Node.js → back-end, IoT, etc.).

Variáveis

- **Declaração de variáveis**
  - **var** → escopo de função, sofre hoisting.
  - **let** → escopo de bloco (recomendado).
  - **const** → valores constantes (imutáveis em referência, mas objetos podem ser alterados)

```
var nome = "Maria";     // escopo função
let idade = 25;         // escopo bloco
const PI = 3.1415;      // constante
```

Escopo

- Em JavaScript, o escopo (scope) determina a visibilidade e acessibilidade de variáveis e funções num determinado contexto de código.

- **Tipos de Escopo**
  - **Escopo Global:** Variáveis e funções declaradas fora de qualquer bloco ou função são globais e podem ser acessadas em qualquer parte do código.
  - **Escopo de Função (ou Escopo Local):** Variáveis e funções declaradas dentro de uma função só são acessíveis dentro dessa função, ou em escopos internos a ela. O var tem este tipo de escopo.
  - **Escopo de Bloco:** Introduzido com let e const (no ES6), este escopo restringe a variável ao bloco onde foi declarada (entre chaves {}), como um if ou um loop for. Variáveis var declaradas dentro de um bloco não são restritas a ele, sendo acessíveis globalmente

Escopo

```
if (true) {
 var x = 10; // disponível fora do bloco
 let y = 20; // restrito ao bloco
}
console.log(x); // 10
console.log(y); // erro!
```

Como o Escopo Funciona

- **Hierarquia:** Escopos internos (ou "filhos") podem aceder variáveis de escopos externos (ou "pais"), mas o contrário não é possível. Uma variável local de uma função não é acessível fora dela.

- **Escopo Léxico (ou Estático):** O escopo de uma variável é determinado no momento da sua declaração, não no momento em que é invocada. Isso significa que uma função terá acesso às variáveis declaradas no seu próprio escopo e nos escopos onde foi definida.

Boas Práticas

- **Evite o Escopo Global Excessivo:** Variáveis globais podem ser acedidas e modificadas em qualquer lugar, levando a erros difíceis de identificar. É preferível confiná-las a escopos menores.

- **Use let e const:** Prefira let e const em vez de var, pois elas introduzem o escopo de bloco, proporcionando um controlo mais granular sobre a visibilidade das variáveis e evitando potenciais problemas.

Tipos de Dados

- **Primitivos**
  - **string:** "Olá", 'Mundo', `template`
  - **number:** 42, 3.14
  - **boolean:** true, false
  - **null:** ausência intencional
  - **undefined:** valor não definido
  - **symbol:** identificador único
  - **bigint:** inteiros grandes (123n)

Tipos de Dados

- **Estruturados**
  - Object
  - Array
  - Function

```
let aluno = {nome: "João", idade: 20};

let lista = [1, 2, 3];
```

Operadores

- Em JavaScript, operadores são símbolos que realizam ações em valores (operandos), realizando desde cálculos aritméticos, comparações lógicas e atribuição de valores, até operações de concatenação de strings.

Operador	Nome	Exemplo	Resultado
==	Igual	a == b	Verdadeiro se a for igual a b
!=	Diferente	a != b	Verdadeiro se a não for igual a b
<>	Diferente	a <> b	Verdadeiro se a não for igual a b
===	Idêntico	a === b	Verdadeiro se a for igual a b e for do mesmo tipo
!==	Não idêntico	a !== b	Verdadeiro se a não for igual a b, ou eles não são do mesmo tipo
<	Menor que	a < b	Verdadeiro se a for menor que b
>	Maior que	a > b	Verdadeiro se a for maior que b
<=	Menor ou igual	a <= b	Verdadeiro se a for menor ou igual a b.
>=	Maior ou igual	a >= b	Verdadeiro se a for maior ou igual a b.

Operadores Aritméticos

- Realizam operações matemáticas.
  - + (Adição): Soma dois números.
  - - (Subtração): Subtrai um número de outro.
  - * (Multiplicação): Multiplica dois números.
  - / (Divisão): Divide um número por outro.
  - % (Módulo): Retorna o resto da divisão.
  - ** (Exponenciação): Eleva um número a uma potência.
  - ++ (Incremento): Adiciona um ao operando (ex: x++).
  - -- (Decremento): Subtrai um do operando (ex: x--).

```
let a = 5, b = 2;

console.log(a + b);

// 7

console.log(a ** b);

// 25
```

Operadores de Comparação

- Comparam dois valores e retornam um booleano (verdadeiro ou falso).
  - == (Igualdade Abstrata): Retorna true se os valores são iguais após a conversão de tipo.
  - === (Igualdade Estrita): Retorna true se os valores e os tipos são idênticos.
  - != (Diferença Abstrata): O oposto de ==.
  - !== (Diferença Estrita): O oposto de ===.
  - > (Maior que), < (Menor que), >= (Maior ou igual a), <= (Menor ou igual a).

```
5 == "5"   // true
5 === "5"  // false
```

Operadores Lógicos

- Usados para combinar valores booleanos (verdadeiro ou falso).
  - && (E Lógico): Retorna true apenas se ambos os operandos forem verdadeiros.
  - || (OU Lógico): Retorna true se pelo menos um dos operandos for verdadeiro.
  - ! (NÃO Lógico): Inverte o valor booleano do operando (se for true, retorna false, e vice-versa).

```
let idade = 18;

console.log(idade >= 18 && idade < 60);
```

Operadores de Atribuição

- Atribuem um valor a uma variável.
  - = (Atribuição): Atribui o valor da direita à variável da esquerda (ex: x = 10).
  - += (Atribuição de Adição): Adiciona um valor ao operando e atribui o resultado (ex: x += 5 é x = x + 5).
  - Outros como -=, *=, /= funcionam de forma semelhante para subtração, multiplicação e divisão, respetivamente.

Outros Operadores

- Concatenação de Strings: O operador + pode ser usado para juntar strings (ex: "Olá" + " " + "Mundo" resulta em "Olá Mundo").

- Acesso a Propriedades: O ponto (.) é usado para acessar propriedades de objetos (ex: meuObjeto.nome).

Funções

- **Definição:** blocos de código reutilizáveis que realizam uma tarefa específica ou retornam um valor.

- **Motivação:** evitam repetição de código, facilitam manutenção e organização.

- **Analogia:** como uma "receita" que recebe ingredientes (parâmetros), executa passos (código) e retorna um prato pronto (resultado).

Declaração de Funções Clássicas

- Possuem nome (soma).
- Podem ser chamadas em qualquer lugar, até antes da declaração, devido ao hoisting.

```
function soma(a, b) {
 return a + b;
}
console.log(soma(2, 3)); // 5
```

Hoisting

- O JavaScript "move" a função declarada para o topo durante a execução.

```
console.log(dobro(4)); // 8

function dobro(x) {
 return x * 2;
}
```

Funções Anônimas

- Não têm nome explícito.
- São atribuídas a variáveis ou passadas como argumento.
- Não sofrem hoisting (não podem ser chamadas antes de declaradas).

```
let dobro = function(x) { return x * 2; };

console.log(dobro(4)); // 8
```

Exemplo em eventos

```
document.getElementById("btn").onclick = function() {
 alert("Botão clicado!");
};
```

Arrow Functions (ES6+)

- Sintaxe mais curta.
- O return é implícito se tiver apenas uma instrução.
- Não têm this próprio → herdam o this do contexto externo.

```
let quadrado = x => x * x;

console.log(quadrado(5)); // 25
```

Exemplo com array

```
let numeros = [1, 2, 3, 4];

let pares = numeros.filter(n => n % 2 === 0);

console.log(pares); // [2, 4]
```

Parâmetros Padrão

- Permitem definir valores default para parâmetros.
- Útil para evitar undefined.

```
function saudacao(nome = "Visitante") {
 console.log("Olá " + nome);
}
saudacao(); // "Olá Visitante"
saudacao("Maria"); // "Olá Maria"
```

Funções como Primeira Classe

- Em JavaScript, funções são First-Class Citizens:
  - Podem ser atribuídas a variáveis.
  - Passadas como argumento para outras funções.
  - Retornadas por outras funções.

```
function executar(funcao, valor) {
 return funcao(valor);
}
let triplo = x => x * 3;
console.log(executar(triplo, 5)); // 15
```

Funções de Ordem Superior

- Definição: funções que recebem ou retornam outras funções.
- Exemplos clássicos: map, filter, reduce.

```
let numeros = [1, 2, 3, 4, 5];
let dobrados = numeros.map(n => n * 2);
console.log(dobrados); // [2, 4, 6, 8, 10]
```

Diferença entre function e arrow function no this

- Função normal: this depende de como é chamada.
- Arrow function: this é léxico (vem do contexto externo).

```
let obj = {
 nome: "JS",
 normal: function() { console.log(this.nome); },
 arrow: () => console.log(this.nome)
};

obj.normal(); // "JS"
obj.arrow();  // undefined (herda o this do escopo global)
```

Boas Práticas

- Prefira arrow functions para funções pequenas e callbacks.
- Use funções nomeadas quando quiser mais clareza no debug (stack trace).
- Sempre documente os parâmetros e valores de retorno.
- Evite funções muito longas: aplique Single Responsibility Principle.

DOM (Document Object Model)

- O DOM (Document Object Model) é a representação estruturada da página HTML, permitindo que o JavaScript acesse e modifique seus elementos.
- Representação da página HTML como árvore.
- Cada tag = nó (elemento, atributo ou texto).
- Podemos buscar, criar e alterar elementos.

Seletores de Elementos

- getElementById()
- Seleciona um único elemento pelo seu id.
- Sempre retorna um único nó (ou null se não existir).

```
<h1 id="titulo">Título Original</h1>
<script>
 let titulo = document.getElementById( "titulo");
 console.log(titulo.innerText);  // "Título Original"
</script>
```

Seletores de Elementos

- getElementsByClassName()
- Seleciona elementos que possuem determinada classe CSS.
- Retorna uma HTMLCollection (parecida com array, mas não exatamente).

```
<ul>
 <li class="item">Item 1</li>
 <li class="item">Item 2</li>
</ul>
<script>
 let itens = document.getElementsByClassName( "item");
 console.log(itens[ 0].innerText); // "Item 1"
</script>
```

Seletores de Elementos

- querySelector() e querySelectorAll()
- querySelector → retorna o primeiro elemento que corresponde ao seletor CSS.
- querySelectorAll → retorna todos os elementos que correspondem ao seletor CSS, em um NodeList (iterável com forEach).

Seletores de Elementos

```
<p>Primeiro</p>
<p>Segundo</p>
<script>
 let primeiro = document.querySelector("p");
 let todos = document.querySelectorAll("p");
 console.log(primeiro.innerText); // "Primeiro"
 todos.forEach(p => console.log(p.innerText));
 // "Primeiro", "Segundo"
</script>
```

Dica: `querySelector` é mais flexível porque aceita qualquer seletor CSS (`#id`, `.classe`, `tag`, combinações).

Alteração de Conteúdo e Estilo

- innerText
  - Muda apenas o texto visível do elemento.

```
titulo.innerText = "Novo título";
```

Alteração de Conteúdo e Estilo

- innerHTML
  - Permite adicionar HTML dentro do elemento.

```
titulo.innerHTML = "<i>Título em itálico</i>";
```

Atenção: `innerHTML` pode ser perigoso se usado com dados externos → risco de **XSS (Cross-Site Scripting)**.

Alteração de Conteúdo e Estilo

- style
  - Permite alterar propriedades CSS diretamente via JS.

```
titulo.style.color = "blue";

titulo.style.fontSize = "30px";

titulo.style.backgroundColor = "yellow";
```

Criando e Inserindo Elementos

- createElement()
  - Cria um novo elemento HTML na memória (ainda não visível).

```
let novo = document.createElement("li");

novo.innerText = "Item novo";
```

Criando e Inserindo Elementos

- appendChild()
  - Adiciona o novo elemento dentro de outro já existente.

```
document.querySelector("ul").appendChild(novo);
```

Outras Formas de Inserção

- prepend()
  - Insere o elemento no início.

```
document.querySelector("ul").prepend(novo);
```

Outras Formas de Inserção

- insertBefore()
  - Insere o elemento antes de outro.

```
let lista = document.querySelector("ul");
let primeiro = lista.firstChild;
lista.insertBefore(novo, primeiro);
```

Outras Formas de Inserção

- removeChild()
  - Remove um elemento existente.

```
lista.removeChild(primeiro);
```

Alterando Atributos

```
<img id="foto" src="foto1.jpg" alt="Imagem">
<script>
 let img = document.getElementById("foto");
 img.setAttribute("src", "foto2.jpg");
 img.setAttribute("alt", "Nova imagem");
 console.log(img.getAttribute("src")); // "foto2.jpg"
</script>
```

Classes e Estilos Dinâmicos

- classList
  - Manipula classes CSS dinamicamente:

```
titulo.classList.add("destaque");      // adiciona classe
titulo.classList.remove("destaque");   // remove classe
titulo.classList.toggle("ativo");      // alterna
titulo.classList.contains("ativo");    // verifica se existe
```

Muito útil para **animações e interações** com CSS.

Exemplo Completo

HTML

```html
<h1 id="titulo">Lista de Compras</h1>
<ul id="lista">
 <li class="item">Arroz</li>
 <li class="item">Feijão</li>
</ul>
```

JS

```html
<script>
 let titulo = document.getElementById( "titulo");
 titulo.innerText = "Minha Lista";
 titulo.style.color = "green";

 let lista = document.getElementById( "lista");
 let novoItem = document.createElement( "li");
 novoItem.innerText = "Macarrão";
 lista.appendChild(novoItem);

 // Alterando a classe dinamicamente
 novoItem.classList.add( "destacado");
</script>
```

Eventos

- Ações que o usuário ou navegador disparam (clique, teclado, carregamento, etc.).

```
<button id="btn">Clique aqui</button>
<script>
document.getElementById("btn").addEventListener("click", function() {
 alert("Botão clicado!");
});
</script>
```

Outros eventos comuns: mouseover, mouseout: keydown, keyup, submit, change

Exercício - Calculadora de Compras

- Implemente uma aplicação web simples que contenha:
- Campos de entrada para:
  - Nome do item (string)
  - Quantidade (number)
  - Preço unitário (number)
- Botão “Adicionar Item” que insere o item em uma lista.
- A lista deve mostrar cada item no formato:
  - Nome - Quantidade x Preço = Subtotal
- Um botão “Calcular Total” que percorre todos os itens e exibe o valor total da compra.
- Um botão “Limpar Lista” que esvazia a lista.
- Estilização dinâmica:
  - Quando um item for adicionado, ele deve aparecer em verde.
  - Se clicar em um item, ele deve ser removido da lista.

Estrutura sugerida - HTML

```html
<h2>Lista de Compras</h2>

<input id="nome" type="text" placeholder="Nome do item">
<input id="quantidade" type="number" placeholder="Quantidade">
<input id="preco" type="number" placeholder="Preço">
<button id="adicionar">Adicionar Item</button>
<button id="calcular">Calcular Total</button>
<button id="limpar">Limpar Lista</button>
<ul id="lista"></ul>
<h3 id="total">Total: R$ 0.00</h3>
```

Conceitos aplicados

- Variáveis (let, const)
- Tipos de dados (string, number, parseInt/parseFloat)
- Operadores (aritméticos, concatenação)
- Funções (clássicas + anônimas nos eventos)
- DOM (getElementById, createElement, appendChild)
- Eventos (click, remoção ao clicar em item)
- Manipulação de elementos (texto, cor, remoção, total dinâmico)

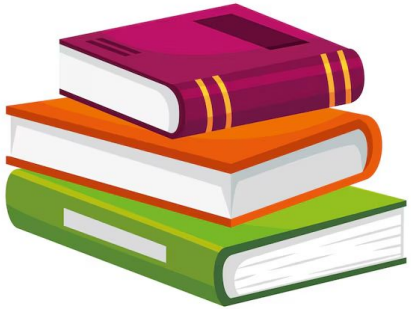

LET'S
CODE

Bibliografia Básica

- LUCKOW, Décio Heinzelmann; MELO, Alexandre Altair de. Programação Java para a Web. São Paulo, SP: Novatec, 2010. 638 p. ISBN 9788575222386.
- GEARY, David; HORSTMANN, Cay. Core JavaServer Faces. 3. ed. Rio de Janeiro, RJ: Alta Books, 2012: ISBN: 9788576086420.
- SILVA, Maurício Samy. HTML 5: a linguagem de marcação que revolucionou a web. 2. ed. São Paulo: Novatec, 2014. 335 p. ISBN 9788575224038.
- FLANAGAN, David. JAVASCRIPT – O Guia Definitivo. Bookman, 6ª ED./2012, 856583719x/97885658371 94.

Bibliografia Complementar

- KURNIAWAN, Budi. Java para a Web com Servlets, JSP e EJB: Budi Kurniawan; tradução Savannah Hartmann; revisão técnica Alfredo Dias da Cunha Júnior. Rio de Janeiro, RJ: Ciência Moderna, 2002. xxiv, 807 p. ISBN 8573932104 (broch.).
- FREEMAN, Elisabeth; FREEMAN, Eric. Use a cabeça!: HTML com CCS e XHTML. 2. ed. Rio de Janeiro, RJ: Alta Books, 2008. xxxi, 580 p. ISBN 9788576082187 (broch.).
- URUBATAN, Rodrigo. Ruby on rails: desenvolvimento fácil e rápido de aplicações Web. São Paulo, SP: Novatec, 2009. 285 p. ISBN 9788575221846 (broch.).
- GONÇALVES, Edson. Desenvolvendo aplicações Web com NetBeans IDE 6. Rio de Janeiro: Ciência Moderna, 2008. 581 p. : CD-ROM ISBN 97885739366742.
- BASHAM, Bryan. Use a cabeça!: Servlets & JSP. 2. ed. Rio de Janeiro, RJ: Alta Books, 2008. ISBN 9788576082941.



Obrigado!

Dúvidas?

Universidade Federal do Ceará - *Campus* Quixadá

Prof. Francisco Victor da Silva Pinheiro
victorpinheiro@ufc.br