Fundamentos de JavaScript

Leandro M. Tosta

Universidade Estadual de Londrina (UEL)

7 de outubro de 2025

Sumário

Introdução

Todos os exemplos práticos, estão disponíveis neste repositório.

- https://github.com/iRocktys/Introduction-JavaScript

Introdução

JavaScript é uma linguagem de programação de alto nível, interpretada e dinâmica. Em termos simples, ela permite que você crie páginas web interativas, animações, jogos e etc.

- JavaScript foi criado para rodar nos navegadores web. Hoje, graças a ambientes como o Node.js, ele também pode ser executado em servidores.
- A forma mais comum de usar JavaScript em um projeto é criando um arquivo externo (.js) e linkando-o no seu arquivo HTML.

Inserindo JavaScript em uma página HTML.

Exemplo Prático 1

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8"/>
    <title>Minha Primeira Pgina JS</title>
</head>
<body>
    <h1>Bem-vindos  Aula de JS!</h1>
    <script src="script.js"></script>
</body>
</html>
```

Fundamentos da Linguagem

Para trabalhar com dados em JavaScript, precisamos conhecer os tipos primitivos que a linguagem utiliza para armazenar a informação.

Tipos de dados

- **string:** Textos
- **number:** Números
- **boolean:** Valores lógicos

Valores Especiais

- **null:** Variável sem valor
- **undefined:** Variável que nunca foi inicializada

Fundamentos da Linguagem

Em JavaScript, podemos declarar variáveis de três formas:

- **var:** Tem um escopo mais "solto".
- **let:** Declara variáveis que podem ter seu valor alterado.
- **const:** Declara variáveis que não podem ser reatribuídas.

Exemplo Prático 2

Criar um código que armazena informações pessoais básicas.

Operadores são símbolos que realizam alguma operação em um ou mais valores. No JavaScript, é vital entender como os operadores de comparação lidam com os diferentes tipos de dados.

Tabela 1: Operadores Aritméticos e de Comparação em JS.

Categoria	Operador	Significado	Exemplo	Resultado
Aritméticos	`+`	Adição/Concatenação	`5 + 3`	8
	`-`	Subtração	`10 - 5`	5
	`*`	Multiplicação	`3 * 4`	12
	`/`	Divisão	`10 / 2`	5
Comparação	`==`	Igualdade (Valor)	`10 == '10'`	`true`
	`===`	Igualdade Estrita	`10 === '10'`	`false`
	`!=`	Diferente (Valor)	`10 != 10`	`false`
	`>` / `<`	Maior/Menor que	`7 > 5`	`true`

Tabela 2: Operadores Lógicos e Funções Matemáticas Essenciais.

Função		Significado	Exemplo	Result.
Lógicos	`&&`	E (AND)	`true && false`	`false`
	`||`	OU (OR)	`true || false`	`true`
	`!`	NÃO (NOT)	`!true`	`false`
Funções Math	`.floor()`	Arredonda para baixo	`Math.floor(4.9)`	4
	`.ceil()`	Arredonda para cima	`Math.ceil(4.1)`	5
	`.random()`	Gera decimal aleatório (0 a 1)	`Math.random()`	*Ex.: 0.873*
	`.max()`	Retorna o maior valor	`Math.max(10, 20)`	20
	`.min()`	Retorna o menor valor	`Math.min(10, 5)`	5
	`.pow(b, e)`	Base `b` elevada ao expoente `e`	`Math.pow(2, 3)`	8
	`.sqrt()`	Retorna a raiz quadrada	`Math.sqrt(9)`	3
	`.abs()`	Retorna o valor absoluto	`Math.abs(-5)`	5

Exemplo Prático 3

Vamos explorar a diferença entre == e ===.

Controle de Fluxo

O controle de fluxo permite que seu código tome decisões.

- **Estruturas condicionais:** Permitem executar blocos de código com base em condições.
- **switch:** Uma alternativa ao ***(if...else if)*** para múltiplos casos.
- **Operador ternário:** Uma forma mais curta de um ***(if...else)*** simples.

Exemplo Prático 4

Vamos explorar a diferença entre as estruturas condicionais.

Exercício 1

Crie um código que verifica a idade de uma pessoa para decidir se ela pode dirigir.

Estruturas de Repetição

As estruturas de repetição (ou laços) nos permitem executar um bloco de código várias vezes.

- **for:** Ótimo para quando você sabe quantas vezes precisar repetir.
- **while:** Repete enquanto uma condição for verdadeira.
- **for...of:** Ideal para iterar sobre os valores de coleções iteráveis, como Arrays e Strings.
- **for...in:** Ideal para iterar sobre as propriedades de Objetos e Arrays.

Exemplo Prático 5

Laços de repetição **for** e **while**.

Exercício 2

Crie um programa que gere uma senha aleatória de 10 caracteres. Para isso, defina uma string com os caracteres permitidos (letras e números, por exemplo). Em seguida, use um laço for para repetir 10 vezes o processo de: extrair um caractere aleatório dessa string base (usando Math.random) e concatená-lo a uma variável de senhaGerada. Exiba a senha final no console.

DICA: A função abaixo gera um índice aleatório que está dentro dos limites da string.

```
const indiceAleatorio = Math.floor(Math.random() *
caracteresBase.length);
```

Funções

Funções são blocos de código reutilizáveis que nos permitem organizar, nomear e executar tarefas específicas.

Categoria	Sintaxe (Exemplo)	Quando Usar
Clássica	`function nome() {...}`	Para funções que podem ser chamadas antes de serem definidas no código.
Expressão/Anônima	`const nome = function() {...}`	Quando a função precisa ser passada como valor para outra variável ou parâmetro.
Arrow Functions	`const nome = () => {...}`	Para sintaxe curta, especialmente em `callbacks`.
Funções de Ordem Superior	`function f1(f2) {...}`	Funções que recebem outras funções (`callbacks`) ou retornam uma nova função (`closures`).
Assíncrona (`async/await`)	`async function nome() {...}`	Para trabalhar com operações demoradas (ex: requisições de rede) de forma não-bloqueante.

Exemplo Prático 6

Explorar a diferença entre os tipos de funções.

Exercício 3

Sua missão é criar um sistema de cálculo que aplique um desconto várias vezes, mas sem usar recursão. Você precisará de duas funções que trabalham juntas:

1. Crie uma Função de **Expressão (Anônima)** chamada **aplicarDesconto** que aceite o valor atual e a taxa de desconto. Ela deve retornar o valor final após o desconto.
2. Crie uma Função de **Ordem Superior (Declaração)** chamada **calcularEmLote** que aceite três parâmetros: o valor inicial do produto, o número de vezes que o desconto deve ser aplicado e a função de callback (**aplicarDesconto**).
3. Dentro de **calcularEmLote**, use um laço **for** para repetir o processo de cálculo. A cada iteração, chame o callback (**aplicarDesconto**), atualize o valor do produto com o resultado e continue até o número de vezes do lote ser atingido. A função deve retornar o valor final.

Ao final, use o sistema para calcular o preço final de um produto de **R$ 500,00** após aplicar um desconto de **10% por 3 vezes seguidas**.

Arrays e Objetos

Arrays são a estrutura de dados mais fundamental para lidar com listas ordenadas no JavaScript.

Tabela 3: Métodos Mutáveis (Modificam o Array Original).

Método	**Propósito**	**Sintaxe (Exemplo)**
`.push()`	Adiciona um ou mais elementos ao final.	`array.push(novoItem)`
`.pop()`	Remove e retorna o último elemento.	`array.pop()`
`.shift()`	Remove e retorna o primeiro elemento.	`array.shift()`
`.unshift()`	Adiciona elementos ao início.	`array.unshift(novoItem)`
`.reverse()`	Inverte a ordem dos elementos do Array.	`array.reverse()`

Tabela 4: Métodos Imutáveis (Criando um Novo Array).

Método	Propósito	Sintaxe (Exemplo)
`.map()`	Cria um novo Array com o resultado da função.	`array.map(item => item * 2)`
`.filter()`	Cria um novo Array com elementos que passam no teste.	`array.filter(item => item > 10)`
`.reduce()`	Aplica uma função acumuladora para reduzir o Array a um valor.	`array.reduce(acc, item => ...)`
`.forEach()`	Executa uma função para cada elemento do Array (sem retornar valor).	`array.forEach(item => console.log(item))`
`.concat()`	Junta dois ou mais arrays, retornando um novo Array.	`array.concat(array2, array3)`

Exemplo Prático 7

Vamos demonstrar a diferença entre o uso de métodos mutáveis e imutáveis na manipulação de arrays.

Exercício 4

Imagine que você está gerenciando a lista de presença de uma sala de aula no JavaScript. A lista inicial é a seguinte: [**'Alice'**, **'Bruno'**, **'Carlos'**, **'Diana'**]. Sua tarefa é executar uma série de modificações nesta lista, usando os métodos de Array adequados:

- O aluno **"Bruno"** precisou sair, então ele deve ser removido da lista.
- Um novo aluno chamado **"Eva"** acabou de chegar e deve ser adicionado ao final da lista.
- Crie uma nova lista contendo apenas os nomes que têm **5 letras ou menos** (o novo nome, **"Eva"**, deve ser incluído).

Arrays e Objetos

Objetos são o pilar do JavaScript, servindo como "recipientes"para dados relacionados. Para trabalhar com eles de forma eficiente, focamos em três ações principais: **acessar/modificar**, **copiar/mesclar** e **iterar** suas chaves.

Tabela 5: Métodos de Objeto e Retorno.

Método/Operador	**Retorna**
`Spread Operator (...)`	Um novo objeto (cópia rasa).
`Object.keys()`	Um Array de strings contendo as chaves do objeto.
`Object.values()`	Um Array contendo os valores das propriedades do objeto.
`Object.entries()`	Um Array de Arrays, onde cada sub-array é um par [chave, valor].

Exemplo Prático 8

Clonagem, Mesclagem e Iteração de Objetos.

Exercício 5

Sua missão é criar uma função chamada **calcularValorInventario** que recebe dois objetos: o **inventarioBase** e as **atualizacoes**. Dentro da função, primeiro você deve mesclar os dois objetos em um novo objeto chamado **inventarioFinal** (usando o **Spread Operator** ...), garantindo que os dados de **atualizacoes** prevaleçam. Em seguida, itere sobre o **inventarioFinal** (usando **Object.keys**) e calcule o valor total de todos os produtos (`preço * quantidade`) somados. A função deve retornar o valor total em estoque.

Document Object Model

O Document Object Model (DOM) é a representação em árvore da sua página HTML no navegador. O JavaScript usa essa estrutura hierárquica para se conectar aos elementos e manipulá-los, dando vida e interatividade à interface.

Selecionar Elementos

- **document.getElementById('id-do-elemento')**: O método mais rápido. Busca um único elemento através do seu ID.
- **document.querySelector('seletor-css')**: Retorna o **primeiro** elemento que corresponde ao seletor CSS fornecido (ex: 'h1', '.classe', 'id').
- **document.querySelectorAll('seletor-css')**: Retorna uma **NodeList** (que é como um Array) com **todos** os elementos que correspondem ao seletor.

Manipular Conteúdo e Atributos

- **elemento.textContent**: Define ou obtém o conteúdo de texto puro.
- **elemento.innerHTML**: Define ou obtém o conteúdo, permitindo que você injete **código HTML** (use com cautela!).
- **elemento.getAttribute('nome-atributo')**: Obtém o valor de um atributo (ex: src, href, class).
- **elemento.setAttribute('nome-atributo', 'novo-valor')**: Define ou altera o valor de um atributo HTML.

Document Object Model

Manipular Classes e Estilos

- **elemento.classList**: Essencial para gerenciar as classes CSS.
- **elemento.classList.add('nome-classe')**: Adiciona uma nova classe.
- **elemento.classList.remove('nome-classe')**: Remove uma classe.
- **elemento.classList.toggle('nome-classe')**: Adiciona a classe se não existir e remove se existir.
- **elemento.style.propriedadeCSS**: Define um estilo CSS diretamente no elemento.

Trabalhar com Eventos

- **elemento.addEventListener('evento', funcao_callback)**: Anexa uma função que será executada quando o evento especificado ocorrer. O evento mais comum é o 'click'.

Criar, Adicionar e Remover Elementos

- **document.createElement('tag')**: Cria um novo elemento HTML na memória do navegador.
- **elementoPai.appendChild(elementoFilho)**: Adiciona o novo elemento como o último filho de um elemento pai existente.
- **elementoPai.removeChild(elementoFilho)**: Remove um elemento filho da página.

Exemplo Prático 9

O objetivo é criar um contador de clicks com dois botões: um para aumentar e outro para diminuir o valor.

Exercício 6

Crie uma página HTML simples com um título e um botão. Usando apenas **JavaScript**, sua tarefa é:

- **Implementar a Alternância Inicial**: O título deve iniciar com o status **"ATIVO"**.
- **Configurar o Evento Click**: Faça com que o botão alterne o estado do título a cada clique. A alternância deve ser:
  - **Status ATIVO → INATIVO**: Mude o texto para **"INATIVO"** e o atributo **data-status** para **"off"**.
  - **Status INATIVO → ATIVO**: Mude o texto para **"ATIVO"** e o atributo para **"on"**.