Lógica de Programação - JavaScript

Aula 01

Rafael Neri

Introdução

Algoritmos

Um algoritmo é um conjunto de instruções finitas e bem definidas para realizar uma tarefa ou resolver um problema, funcionando como uma receita de bolo. Ele recebe uma entrada (input), processa essa entrada através de passos lógicos e produz uma saída (output). Algoritmos são a base da programação de computadores, mas também são usados em sistemas como redes sociais, motores de busca, GPS e finanças.

Exemplo prático

O Sr. Joaquim, dono da tradicional **Farmácia Boa Saúde**, viu a concorrência crescer e decidiu modernizar seus serviços — afinal, medir pressão já não era mais novidade.

Sua nova ideia genial? Oferecer um cálculo personalizado de **IMC** para cada cliente!
Com um caderninho e uma calculadora, ele registrava o peso, a altura e o resultado, criando um histórico digno de um "check-up de bairro".

Agora, o Sr. Joaquim quer automatizar essa façanha:
Implemente um algoritmo (em português) que leia o peso e a altura de um cliente, calcule o IMC e exiba o resultado.

(Se quiser ganhar um desconto na próxima visita, mostre também a faixa do IMC — o Sr. Joaquim adora esses detalhes!)

JavaScript

Breve histórico:

- Criado em 1995 por Brendan Eich (em 10 dias!).
- Linguagem de tipagem fraca e dinâmica.
  Fraca: Permite que a variável mude de tipo.
  Dinâmica: O tipo é definido em tempo de execução, e não na declaração.
- Interpreta código linha a linha.
- Executa no navegador ou em ambientes como Node.js.

Sistemas de tipos

O sistema de tipos define como os valores são classificados, comparados e manipulados dentro da linguagem.

O JavaScript é uma linguagem de tipagem **fraca** e **dinâmica**, e isso tem implicações importantes.

Tipagem Dinâmica

Significa que o tipo de uma variável é definido em tempo de execução, e pode mudar a qualquer momento.

```
let valor = 10; // tipo: number
console.log(typeof valor); // "number"

valor = "dez"; // tipo muda para string
console.log(typeof valor); // "string"
```

O interpretador JavaScript decide o tipo com base no valor no momento da execução, e não na declaração.

Tipagem Fraca

Em linguagens de tipagem fraca, valores de tipos diferentes podem ser combinados automaticamente — às vezes com resultados inesperados.

```
let resultado = "5" + 3;
console.log(resultado); // "53" — concatenou, não somou!
```

Isso acontece porque o JavaScript converte o número 3 em string antes da operação (coerção implícita).

Outro exemplo curioso:

```
console.log(1 + "1"); // "11"
console.log(1 - "1"); // 0
```

O operador `+` tenta concatenar se houver string, mas o `-` força coerção para número — resultado: comportamentos inconsistentes.

Tipos Primitivos

O JavaScript tem sete tipos primitivos, que são imutáveis e não-estruturados.

Tipo	Exemplo	Descrição
string	`"Farmácia"`	Texto
number	`42`, `3.14`	Número (inteiro ou decimal)
boolean	`true`, `false`	Valor lógico
undefined	—	Variável declarada, mas sem valor
null	—	Ausência intencional de valor
symbol	`Symbol("id")`	Identificador único (usado em objetos)
bigint	`123n`	Números inteiros muito grandes

Tipos de Referência (Objetos)

Além dos tipos primitivos, há os tipos de referência, usados para estruturas de dados mais complexas.

Tipo	Exemplo
Object	`{ nome: "Joaquim", idade: 54 }`
Array	`[10, 20, 30]`
Function	`function calcularIMC(p, a) { ... }`
Date	`new Date()`

Esses tipos armazenam referências na memória, não valores diretamente.

Constantes, Variáveis e Estados

Conceitos:

- Variável: Valor que pode ser alterado (reatribuído). let (uso moderno, escopo de bloco).
- Constante: Valor fixo, não pode ser reatribuído.
- Estado: Conjunto de valores armazenados nas variáveis em um determinado momento. A mudança de estado é o que faz o programa "fazer" coisas.

`var` — o jeito antigo (e perigoso)

- Introduzida desde as primeiras versões da linguagem.
- Escopo de função (ou global, se declarada fora de uma função).
- Permite redeclaração e reatribuição.
- Sofre hoisting — é “içada” para o topo do escopo, mas sem valor inicial.

Resumo:

- Válida dentro de toda a função.
- Evite usar — pode causar comportamentos confusos e bugs sutis.

`let` — o novo padrão (seguro e previsível)

- Introduzido no ES6 (2015).
- Escopo de bloco (limitada a `{}` onde foi declarada).
- Permite reatribuição, mas não redeclaração no mesmo escopo.
- Mais previsível e seguro que `var`.

Resumo:

- Boa escolha para valores que podem mudar.
- Não pode ser redeclarada no mesmo escopo.

`const` — valores fixos e confiáveis

- Também introduzido no ES6.
- Escopo de bloco, como `let`.
- Não pode ser reatribuída — o valor é constante.
- Ideal para valores que não devem mudar (ex.: PI, URLs, configurações).
- Objetos e arrays declarados com `const` ainda podem ser modificados internamente (só não podem ser reatribuídos).

Comparativo:

Característica	var	let	const
Escopo	Função ou global	Bloco	Bloco
Hoisting	Sim (sem valor inicial)	Sim (mas inacessível antes da linha)	Sim (mas inacessível antes da linha)
Redeclaração	Permitida	Não	Não
Reatribuição	Permitida	Permitida	Não
Melhor prática	Evite	Use para valores mutáveis	Use para valores fixos

Regra de ouro:

- Use `const` sempre que possível.
- Só use `let` quando o valor precisar mudar.
- E use `var` apenas se estiver lendo código antigo — ou se quiser se lembrar por que ela caiu em desuso ^^.

Funções Nativas, Coerções e Operadores

Principais funções nativas:

- `prompt()` → entrada de dados
- `console.log()` → saída
- `Number()`, `String()` → conversões (coerções)
- Operadores aritméticos: `+`, `-`, `*`, `/`, `%`, `**`

Quebra de Fluxo e Tomada de Decisão com JavaScript

Todo programa executa instruções em sequência, linha a linha.
Mas, às vezes, queremos que o código “pense” e escolha um caminho.

Rafael Neri

Condicionais `if / else`

```
if (condição) {
  // executa se condição for verdadeira
} else {
  // executa se for falsa
}
```

Condicional `switch`

Usada quando há múltiplas opções possíveis para uma mesma variável.

```
switch (key) {
  case value:
    break;

  default:
    break;
}
```

Operadores Lógicos em JavaScript

Os operadores lógicos permitem combinar condições e controlar decisões do programa.

- AND — `&&`

Retorna `true` somente se **todas** as condições forem verdadeiras.

```
true && true   // true
true && false  // false
```

Operadores Lógicos em JavaScript

- OR — `||`

Retorna `true` se **pelo menos uma** condição for verdadeira.

```
true || false  // true
false || false // false
```

Operadores Lógicos em JavaScript

- NOT — `!`

Inverte o valor lógico.

```
!true  // false
!false // true
```

Condicional Ternário (`?` `:`)

Forma curta de um if/else simples.

```
condição ? valorSeVerdadeiro : valorSeFalso;
```

Conceito de Truthy / Falsy

Em JavaScript, qualquer valor pode ser avaliado como verdadeiro (**truthy**) ou falso (**falsy**).

Valores **falsy**:

`false`, `0`, `""`, `null`, `undefined`, `NaN`

Tudo o resto é **truthy**.

Operadores de Coalescência (`??` e `?.`)

`??` — Nullish Coalescing

Retorna o primeiro valor não nulo nem indefinido.

```
let nomeCliente = null;
let nomePadrao = "Cliente desconhecido";
console.log(nomeCliente ?? nomePadrao);
```

`?.` — Optional Chaining

Evita erro quando acessamos propriedades que podem não existir.

```
let cliente = { nome: "Ana" };
console.log(cliente.endereco?.cidade); // undefined (sem erro)
```

Exercícios