Lógica de Programação - JavaScript

Aula 03

Rafael Neri

Revisão dos Conceitos Básicos de JavaScript

Rafael Neri

Tipos

- **Primitivos:** `string`, `number`, `boolean`, `null`, `undefined`, `symbol`, `bigint`.
- **Referência/objetos:** `Object`, `Array`, `Function`, `Date`.

```
let s = "texto";     // string
let n = 42;          // number (integer)
let f = 50.2;        // number (decimal)
let b = true;        // boolean
let a = [1,2,3];     // object (array)
let d = new Date(); // object (date)
```

Variáveis, Constantes e Estado

- `let` para valores que mudam;
- `const` para valores que não serão reatribuídos.
- `var` existe, mas evite (escopo confuso).

Estado = valores atuais das variáveis; alterações de estado mudam o comportamento do programa.

```
const PI = 3.14;

let contador = 0; // estado inicial: 0
contador = contador + 1; // estado atual: 1
```

Coerções

- Em JavaScript existem duas formas de conversão entre tipos:
  - **Coerção implícita** (automática) — o interpretador converte sem você pedir.
  - **Coerção explícita** (conversão direta) — você pede a conversão usando funções/operadores.

Coerção implícita (automática)

- O JS tenta adivinhar o que fazer quando combina valores de tipos diferentes.
- Exemplos comuns:
  - `"5" + 3` → `"53"` (concatenação porque `+` com string converte o número para string).
  - `"5" - 3` → `2` (o `-` força conversão para number).
  - Em contextos booleanos, valores são convertidos para true/false (truthy/falsy).

```
"5" + 3          // "53"
"5" - 3          // 2
Boolean("")      // false
Boolean("texto") // true
```

Coerção explícita (conversões que você chama)

- Use funções/operadores para converter de forma clara e segura:
  - `Number(value)` → tenta converter para número (pode gerar `NaN`).
  - `String(value)` → converte para string.
  - `Boolean(value)` → converte para true/false.
  - `parseInt(str, base)` / `parseFloat(str)` para conversões de strings numéricas.

```
Number("5") + 3          // 8
String(10) + " pessoas" // "10 pessoas"
Boolean(0)               // false
Number("abc")            // NaN (cuidado)
```

Vetores (Arrays)

- Lista ordenada de valores. Acesso por índice (começando em 0).

```
const v = [10, 20, 30];

console.log(v[0]); // 10
```

Matrizes (arrays de arrays)

- Tabela de números ou valores; `matriz[i][j]` acessa linha `i`, coluna `j`.

```
const M = [[1,2], [3,4]];

console.log(M[1][0]); // 3
```

Operadores Aritméticos

- `+`, `-`, `*`, `/`, `%` (módulo), `**` (potência)

```
console.log(2 + 3); // 5
console.log(5 % 2); // 1
console.log(2 ** 3); // 8
```

Operadores Lógicos

- `&&` (AND), `||` (OR), `!` (NOT)

```
console.log(true && false); // false
console.log(true || false); // true
console.log(!true); // false
```

Truthy / Falsy

- Valores falsy: `false`, `0`, `""` (string vazia), `null`, `undefined`, `NaN`.
- Tudo o resto é truthy.

```
if ("") console.log('vai rodar?'); // não roda (falsy)
if ("texto") console.log('texto é truthy'); // roda
```

Estruturas Condicionais (if / else / ternário)

```
const nota = 7;

if (nota >= 7) {
  console.log('Aprovado');
} else {
  console.log('Reprovado');
}

const resultado = nota >= 7 ? 'Aprovado' : 'Reprovado';

switch (resultado) {
    case 'Aprovado':
        console.log('Você foi aprovado.');
        break;
    case 'Reprovado':
        console.log('Reprovado');
        break;
}
```

Operadores de Coalescência e Optional Chaining

- `??` retorna o primeiro valor não `null` / `undefined`.
- `?.` evita erro ao acessar propriedade de algo que pode ser `null` / `undefined`.

```
const nome = null;
console.log(nome ?? 'Sem nome'); // 'Sem nome'

const user = { perfil: null };
console.log(user.perfil?.idade); // undefined (sem erro)
```

Estruturas de Repetição — `for`

```
for (let i = 0; i < 3; i++) {
  console.log(i);
}
```

`while` e `do...while`

```
let i = 0;
while (i < 3) {
  console.log(i);
  i++;
}

// Imprime: 0, 1, 2

let j = 0;
do {
  console.log(j);
  j++;
} while (j < 3);

// Imprime: 0, 1, 2, 3
```

`for...in` vs `for...of`

- `for...in` itera sobre chaves (índices/propriedades).
- `for...of` itera sobre os valores arrays ou strings.

```
const obj = { a: 1, b: 2 };
for (let k in obj) {
    console.log(k);
}

// Imprime: a, b

const arr = [10, 20];
for (let v of arr) {
    console.log(v);
}

// Imprime: 10, 20
```

Métodos de Array — adição/remoção

- `push()` adiciona no final.
- `pop()` remove do final.
- `unshift()` adiciona no início.
- `shift()` remove do início.
- `splice` remove/adiciona em posição.

```
const a = [];
a.push(1);      // [1]
a.push(2);      // [1, 2]
a.unshift(0);   // [0, 1, 2]
a.pop();        // [0, 1]
a.splice(1, 1); // [0]
```

Métodos de Busca

- `includes(value)`: verifica se o array contém `value`.
- `indexOf(value)`: retorna o índice da primeira ocorrência de `value`.
- `find(fn)`: por uma função busca o primeiro elemento que satisfaz a condição.
- `findIndex(fn)`: como `find`, mas retorna o índice.

```
['Ana', 'Beto'].includes('Beto') // true

[5, 9, 7, 9].indexOf(9) // 1

const pessoas = [{nome:'Ana', idade:20}, {nome:'Beto',idade:30}];
pessoas.find(p => p.idade >= 30) // { nome: 'Beto', idade: 30 }

pessoas.findIndex(p => p.nome === 'Ana') // 0
```

Métodos de Transformação / Filtragem

- `map(fn)` transforma cada item.
- `filter(fn)` filtra por condição.

```
const nums = [1,2,3];

const dobrados = nums.map(n => n * 2); // [2,4,6]

const pares = nums.filter(n => n % 2 === 0); // [2]
```

`reduce`, `sort`, `reverse`

- `reduce` acumula valores (somar, contar, etc.).
- `sort` ordena.
- `reverse` inverte.

```
const nums = [3,1,2];
const soma = nums.reduce((s, n) => s + n, 0); // 6
nums.sort((a,b) => a - b); // [1,2,3]
nums.reverse(); // [3,2,1]
```

Funções em Javascript

Rafael Neri

Funções e Assinaturas

Funções são blocos de código que executam uma tarefa específica e podem ser reutilizadas. A **assinatura** descreve o nome da função, seus parâmetros e (quando houver) o valor retornado.

```
function saudacao() {
  console.log("Bem-vindo à Farmácia Boa Saúde!");
}

// Assinatura: saudacao() -> void
```

Características

- **Reutilizáveis:** isolam comportamento.
- **Parametrizáveis:** recebem argumentos para variar a execução.
- **Podem retornar** valores com `return`.
- **Primeira-classe:** podem ser atribuídas a variáveis, passadas como parâmetro e retornadas por outras funções.

Funções nativas (da linguagem)

- Conversão: `Number()`, `String()`, `Boolean()`, `parseInt()`.
- Saída/Depuração: `console.log()`.
- Tempo: `setTimeout()` / `setInterval()` (usos básicos).

```
Number('10') // 10
parseInt('11', 10) // 11  (use radix)
console.log('ok');
```

Operações matemáticas com `Math`

`Math` tem funções úteis:

```
Math.PI         // 3.14159...
Math.round(2.7) // 3
Math.max(1,5,3) // 5
Math.sqrt(9)    // 3
Math.random()   // número entre 0 e 1
```

Como criar funções — formas comuns

1. Declaração (hoisting disponível):

```
function somar(a, b) {
  return a + b;
}
```

2. Expressão de função (atribua a `const`):

```
const subtrair = function(a, b) {
  return a - b;
};
```

3. Arrow function (sintaxe curta):

```
const multiplicar = (a, b) => a * b;
```

Parâmetros padrão, rest e `arguments`

```
function saudacao(nome = 'convidado') {
  return `Olá, ${nome}`;
}

function soma(...valores) {
  let total = 0;
  for (let v of valores) total += v;
  return total;
}

function exemploArguments() {
  // 'arguments' existe em funções tradicionais
  console.log(arguments.length);
}
```

Peculiaridades importantes

- **Hoisting:** declarações `function name(){}` são elevadas; expressões (`const f = ...`) não são.
- **Funções como valores:** passe funções como parâmetros (callbacks).
- **Closures:** função lembra variáveis do contexto onde foi criada.
- **Efeitos colaterais:** modificar variáveis externas pode causar bugs — prefira funções puras quando possível.

```
function criaContador() {
  let c = 0;
  return function() { c += 1; return c; };
}

const cont = criaContador();
cont(); // 1
cont(); // 2  (closure lembra 'c')
```

Assinaturas e exemplos práticos

```
function calcularDesconto(valor, percentual) {
  return valor - valor * (percentual / 100);
}

function elevar(base, expoente) {
  return base ** expoente;
}

console.log(calcularDesconto(100, 10)); // 90
console.log(elevar(2, 3)); // 8
```

Dicas e Boas Práticas

- Dê nomes claros (`calcularMedia`, `buscarUsuario`).
- Prefira funções pequenas e com responsabilidade única.
- Evite efeitos colaterais desnecessários; prefira retornar valores.
- Use `const` para funções atribuídas quando não for reatribuir.

Exercício rápido (opcional)

Escreva uma função `media` que receba três notas e retorne a média. Use parâmetros e `return`.

```
// exemplo esperado:
function media(n1, n2, n3) {
  return (n1 + n2 + n3) / 3;
}

console.log(media(8, 7, 9)); // 8
```

Rafael Neri