

Capítulo: Arrays em JavaScript

1. Introdução aos Arrays

Um **array** é uma coleção **ordenada de valores**, onde cada valor é chamado de **elemento**, e cada elemento possui uma **posição numérica** conhecida como **índice**.

Os arrays em JavaScript são **não tipados**, o que significa que podem armazenar diferentes tipos de valores (números, strings, objetos, funções, etc.) no mesmo array.

Além disso, arrays em JavaScript são **baseados em zero**: o primeiro elemento está na posição **0**, e o último elemento está na posição **length - 1**. O índice mais alto possível é $2^{32} - 2$, devido à forma como o JavaScript armazena índices internamente em 32 bits.

Outro aspecto importante é que arrays são **dinâmicos** — eles podem **crescer ou diminuir** conforme elementos são adicionados ou removidos — e podem ser **esparso**s, ou seja, não precisam ter índices contíguos.

Todos os arrays possuem uma propriedade chamada **length**, que representa seu tamanho lógico (um valor maior que o índice do último elemento).

Importante: Arrays em JavaScript são uma forma especializada de **objeto**, e herdam propriedades e métodos definidos em **Array.prototype**, que fornece um conjunto rico de funcionalidades para manipulação.

Exemplo:

```
let frutas = ["maçã", "banana", "laranja"];
console.log(frutas.length); // 3
console.log(frutas[0]); // "maçã"
console.log(typeof frutas); // "object"
```

2. Criando Arrays

Há várias formas de criar arrays em JavaScript:

a) Literais de Array

A forma mais simples é usar **colchetes ([])** com uma lista separada por vírgulas:

```
let cores = ["vermelho", "verde", "azul"];
```

Se houver **vírgulas duplas**, os elementos ausentes são considerados **undefined**, criando um array **esparso**:

```
let numeros = [1, , 3];
console.log(numeros.length); // 3
console.log(numeros[1]); // undefined
```

b) Operador Spread (...)

O operador **spread** expande o conteúdo de um array (ou outro objeto iterável) dentro de outro contexto.

```
let letras = ["a", "b", "c"];
let maisLetras = [...letras, "d", "e"];
console.log(maisLetras); // ["a", "b", "c", "d", "e"]
```

Ele também funciona com **strings**, transformando-as em arrays de caracteres:

```
let palavra = [..."JavaScript"];
console.log(palavra);
// ["J", "a", "v", "a", "S", "c", "r", "i", "p", "t"]
```

c) Construtor Array()

O construtor `Array()` pode ser usado de três formas:

```
let a = new Array();           // Cria array vazio []
let b = new Array(3);          // Cria array com length = 3, sem valores definidos
let c = new Array(1, 2, 3);    // Cria array [1, 2, 3]
```

Cuidado: `new Array(3)` **não cria três elementos `undefined`**, mas sim um array **vazio com tamanho 3**, ou seja, esparsos.

d) Método Array.of()

Cria um novo array com os valores passados como argumentos, **sem comportamento especial** para números:

```
let numeros = Array.of(3);    // [3]
let outros = Array.of(1, 2, 3); // [1, 2, 3]
```

e) Método Array.from()

Cria um novo array a partir de um **objeto iterável** (como uma string ou outro array). Também aceita um **segundo argumento** — uma função de transformação aplicada a cada elemento.

```
let texto = "ABC";
let letras = Array.from(texto);
console.log(letras); // ["A", "B", "C"]

let dobrados = Array.from([1, 2, 3], x => x * 2);
console.log(dobrados); // [2, 4, 6]
```

Esse método é útil para converter **objetos array-like**, como `NodeList`, em arrays reais — facilitando o uso de métodos como `map()`, `filter()`, etc.

Exemplo com DOM:

```
let botoes = document.querySelectorAll("button");
let listaBotoes = Array.from(botoes);
listaBotoes.forEach(b => console.log(b.textContent));
```

3. Lendo e Escrevendo Elementos

Para **acessar** ou **modificar** elementos, usamos o **operador de colchetes** `[]` com um índice.

```
let cores = ["vermelho", "verde", "azul"];
console.log(cores[0]); // "vermelho"
cores[1] = "amarelo";
console.log(cores); // ["vermelho", "amarelo", "azul"]
```

Como arrays são objetos, também é possível adicionar propriedades nomeadas, mas isso **não afeta** o comprimento (`length`):

```
cores.descricao = "Cores primárias";
console.log(cores.length); // ainda é 3
```

4. Arrays Esparsos

Um **array esperso** é aquele em que os índices **não são contíguos**. Isso pode acontecer, por exemplo, ao definir um elemento com índice maior do que o atual:

```
let a = [];
a[100] = "fim";
console.log(a.length); // 101
```

Nesses casos, o array contém “buracos” — índices sem valor definido (`undefined`).

Os arrays esparsos ocupam menos memória, mas geralmente são **mais lentos** e exigem atenção ao iterar.

5. Propriedade `length`

A propriedade `length` indica o tamanho lógico do array. Ela **não representa necessariamente o número de elementos existentes**.

```
let a = [1, 2, 3];
console.log(a.length); // 3
a[10] = 99;
console.log(a.length); // 11
```

É possível **alterar** manualmente `length`. Se o novo valor for **menor** que o atual, os elementos excedentes serão **removidos**:

```
let numeros = [1, 2, 3, 4];
numeros.length = 2;
console.log(numeros); // [1, 2]
```

6. Adicionando e Removendo Elementos

JavaScript fornece métodos específicos para manipulação de elementos:

- `push()` → adiciona um ou mais elementos ao final.
- `pop()` → remove o último elemento.
- `unshift()` → adiciona elementos no início.
- `shift()` → remove o primeiro elemento.

Exemplo:

```
let frutas = ["maçã", "banana"];
frutas.push("laranja");
console.log(frutas); // ["maçã", "banana", "laranja"]

frutas.shift();
console.log(frutas); // ["banana", "laranja"]
```

O operador `delete` pode ser usado, mas ele **não reorganiza os índices**, deixando o array esperso:

```
let x = [10, 20, 30];
delete x[1];
console.log(x); // [10, empty, 30]
console.log(x.length); // 3
```

7. Iterando Arrays

Existem várias formas de percorrer arrays:

a) **for** tradicional

```
let numeros = [1, 2, 3];
for (let i = 0; i < numeros.length; i++) {
  console.log(numeros[i]);
}
```

b) **for...of**

Percorre diretamente os **valores** dos elementos:

```
for (let valor of numeros) {
  console.log(valor);
}
```

c) **forEach()**

Executa uma função callback para cada elemento existente. Não itera sobre elementos ausentes (útil em arrays esparsos).

```
numeros.forEach((valor, indice) => {
  console.log(`Índice ${indice}: ${valor}`);
});
```

8. Arrays Multidimensionais

O JavaScript não possui **arrays verdadeiramente multidimensionais**, mas podemos **simular** esse comportamento com **arrays dentro de arrays**.

Exemplo:

```
let matriz = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];

console.log(matriz[0][1]); // 2
console.log(matriz[2][2]); // 9
```

Também é possível usar loops aninhados para percorrer todas as posições:

```
for (let linha of matriz) {  
  for (let valor of linha) {  
    console.log(valor);  
  }  
}
```

Conclusão

Os **arrays** são uma das estruturas de dados mais importantes do JavaScript. Sua flexibilidade — como o fato de serem dinâmicos, não tipados e herdarem métodos poderosos de `Array.prototype` — os torna fundamentais para manipulação de coleções, listas e dados em geral.

Entender os conceitos de **índices, comprimento, arrays esparsos, métodos de inserção, remoção e iteração** é essencial para dominar a programação em JavaScript e trabalhar eficientemente com o DOM, APIs e dados complexos.

Exercícios - Arrays em JavaScript

1. Criação de Arrays

Crie três arrays de maneiras diferentes:

- Um com **array literal**, contendo nomes de frutas;
- Um com o **construtor `Array()`**, contendo números de 1 a 5;
- Um usando o **método `Array.of()`**, contendo três valores booleanos.

Depois, exiba cada um no console.

2. Usando o Operador Spread

Dado o array:

```
let letras = ["A", "B", "C"];
```

Use o **operador spread (`...`)** para criar um novo array que contenha todas as letras acima e mais as letras "D" e "E". Depois, mostre o novo array no console.

3. Conversão com `Array.from()`

Utilizando o código abaixo:

```
let texto = "JavaScript";
```

Use o método `Array.from()` para transformar essa string em um array de caracteres. Em seguida, exiba cada letra com `forEach()`.

Experimente também criar um segundo array que contenha todas as letras em minúsculas.

4. Acessando e Modificando Elementos

Crie um array chamado `cores` com os valores `["vermelho", "verde", "azul"]`. Depois:

1. Altere o segundo elemento para `"amarelo"`;
2. Adicione `"roxo"` ao final do array;
3. Exiba o array final no console.

5. Trabalhando com Arrays Esparsos

Crie um array vazio chamado `valores`. Atribua o valor `100` ao índice `5` e exiba o resultado no console. Depois, verifique o valor da propriedade `.length`. Explique: por que o tamanho é maior que o número de elementos existentes?

6. Propriedade Length

Dado o código:

```
let numeros = [10, 20, 30, 40, 50];
```

- Exiba o tamanho do array;
- Altere `length` para 3;
- Mostre o novo conteúdo do array no console.

O que aconteceu com os elementos após o índice 2?

7. Inserindo e Removendo Elementos

Crie um array com os valores `["maçã", "banana"]`. Depois:

1. Use `push()` para adicionar `"laranja"`;
2. Use `unshift()` para adicionar `"uva"` no início;
3. Use `pop()` para remover o último elemento;
4. Use `shift()` para remover o primeiro elemento.

Exiba o array a cada etapa para observar as mudanças.

8. Iterando Arrays

Crie um array com os números de 1 a 5. Percorra o array de três maneiras diferentes:

- Com um **loop for tradicional**;
- Com um **loop for...of**;
- Com o método **forEach()**.

Em cada caso, exiba os valores no console.

9. Array Multidimensional

Crie um array bidimensional que represente uma tabela 3x3, contendo números de 1 a 9.

Depois:

- Exiba o número da segunda linha e terceira coluna.
- Use dois loops **for** aninhados para imprimir todos os números da matriz.

Dica: `matriz[1][2]` acessa a linha 2, coluna 3.

10. Arrays e o DOM

Crie uma página HTML simples com **cinco botões**:

```
<button>Botão 1</button>
<button>Botão 2</button>
<button>Botão 3</button>
```

```
<button>Botão 4</button>  
<button>Botão 5</button>
```

No script JavaScript:

1. Use `document.querySelectorAll("button")` para selecionar todos os botões;
2. Converta o resultado em um array usando `Array.from()`;
3. Use `forEach()` para mudar a cor de fundo de cada botão para uma cor diferente.

Exemplo:

```
let botoes = Array.from(document.querySelectorAll("button"));  
let cores = ["red", "green", "blue", "orange", "purple"];  
  
botoes.forEach((btn, i) => {  
  btn.style.backgroundColor = cores[i];  
});
```

Desafio Extra

Crie uma página que gera uma **tabela de multiplicação (10x10)** usando um **array bidimensional** e insere o resultado dinamicamente no DOM com `innerHTML`.