

Arrays em JavaScript

Parte 1

Arrays

Um **array** é uma coleção ordenada de valores, onde cada elemento possui uma posição numérica chamada **índice**.

Os arrays em JavaScript são:

- ~ Não tipados (podem conter diferentes tipos de valores)
- ~ Baseados em zero (o primeiro índice é `0`)
- ~ Dinâmicos (podem crescer ou diminuir conforme necessário)
- ~ Possivelmente esparsos (os índices não precisam ser contínuos)
- ~ Possuem uma propriedade `length`

Arrays e Objetos

Os arrays são uma forma especializada de objeto JavaScript.

Eles herdam propriedades de `Array.prototype`, que define um rico conjunto de métodos de manipulação.

Arrays tipados possuem tamanho fixo e tipo numérico fixo para seus elementos.

Exemplo:

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

Criando Arrays

Criando Arrays

Existem várias formas de criar arrays:

- ↗ Literais de array
- ↗ Operador Spread (...)
- ↗ Construtor `Array()`
- ↗ Métodos fábrica `Array.of()` e `Array.from()`

Array Literals

A forma mais simples de criar um array é usando colchetes `[]` com elementos separados por vírgulas.

```
let frutas = ["maçã", "banana", "laranja"];  
console.log(frutas[1]); // banana
```

Operador Spread (...)

↗ O operador spread espalha os elementos de um array dentro de outro.

```
let a = [1, 2, 3];  
let b = [0, ...a, 4];  
console.log(b); // [0, 1, 2, 3, 4]
```

↗ Também funciona com **strings** (iteráveis):

```
[... 'Olá']; // ["O", "l", "á"]
```

Construtor Array()

↗ Cria arrays de diferentes formas:

```
let vazio = new Array();           // []  
let tamanho = new Array(3);       // [ <3 empty items> ]  
let numeros = new Array(1, 2);    // [1, 2]
```


Array.of()

↗ Cria um novo array utilizando os argumentos passados para função:

```
// Cria um novo array com os valores fornecidos  
let a = Array.of(1, 2, 3); // [1, 2, 3]
```

Array.from()

🌀 Recebe um objeto iterável (arrays, strings). Permite realizar cópias:

```
let copy = Array.from(original);
```

🌀 Aceita um segundo argumento que é uma função. Cada elemento do Array é passado por essa função:

```
// Segundo argumento: função de mapeamento  
let c = Array.from([1, 2, 3], x => x * 2);  
console.log(c); // [2, 4, 6]
```

🌀 Transforma string em array: `let b = Array.from("ABC"); // ["A", "B", "C"]`

Lendo e Escrevendo Elementos

Lendo e Escrevendo Elementos

🌀 Use colchetes `[]` com um índice inteiro não negativo.

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

Lembre-se: arrays são um **tipo especializado de objeto**.

🌀 É importante distinguir um índice de array de uma propriedade de um objeto

```
let cores = [0,1,2,3,4]; //cores.length - indica quantidade de elementos  
let obj = {length:5} //propriedade com valor inteiro  
console.log(cores.length !== obj.length)
```

Arrays Esparsos

Arrays Esparsos

~ Um **array esperso** não tem índices contíguos a partir do zero. Em outras palavras, há espaços no array que são vazios.

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

~ Arrays esparsos são **mais lentos**, mas **mais econômicos em memória**.

~ Se precisar lidar com elementos inexistentes, trate-os como **undefined**.

Propriedade Length

🌀 Todo array tem a propriedade `length`, que é sempre maior que o maior índice existente.

```
let a = [1, 2, 3];  
a.length = 5;  
console.log(a); // [1, 2, 3, <2 empty items>]
```

🌀 Se definir `length` como menor que o valor atual, os elementos são apagados:

```
a.length = 2;  
console.log(a); // [1, 2]
```

Adicionando e Removendo Elementos

Adicionando e Removendo Elementos

Métodos principais:

- `push()` — adiciona no final
- `pop()` — remove o último
- `unshift()` — adiciona no início
- `shift()` — remove o primeiro

```
let frutas = ["maçã"];  
frutas.push("banana"); // ["maçã", "banana"]  
frutas.unshift("uva"); // ["uva", "maçã", "banana"]  
frutas.pop(); // ["uva", "maçã"]  
frutas.shift("uva")
```

Delete e Arrays

Delete e Arrays

🏹 O operador `delete` remove o elemento, mas não ajusta os índices nem altera `length`.

```
let a = ["x", "y", "z"];  
delete a[1];  
console.log(a); // ["x", empty, "z"]  
console.log(a.length); // 3
```

🏹 O array se torna **esparso**.

Iterando Arrays

Iterando Arrays

O `for/of` percorre os elementos em ordem crescente:

```
let cores = ["azul", "verde", "vermelho"];  
for (let cor of cores) console.log(cor);
```

Com `forEach()` :

```
cores.forEach(cor => console.log(cor.toUpperCase()));
```

O `forEach()` ignora elementos ausentes em arrays esparsos.

Arrays Multidimensionais

Arrays Multidimensionais

~ JavaScript não possui arrays verdadeiramente multidimensionais, mas podemos **simular** usando arrays dentro de arrays.

```
let matriz = [  
  [1, 2, 3],  
  [4, 5, 6]  
];  
  
console.log(matriz[1][2]); // 6
```

~ Para acessar o valor de um array multidimensional, como uma matriz, utilizamos o operador `[]` duas vezes: `matriz[0][0]`

Resumo

- Arrays são **objetos especiais** com índices numéricos.
- São **dinâmicos**, **não tipados** e podem ser **esparsos**.
- Possuem a propriedade **length**.
- Podem ser criados de várias formas: `[]` , `Array()` , `Array.of()` , `Array.from()` .
- Métodos principais: `push()` , `pop()` , `shift()` , `unshift()` , `forEach()` .
- Permitem simular **estruturas multidimensionais**.

Desafio

Pratique

Tente criar um script que:

- ↗ Crie um array de nomes.
- ↗ Adicione e remova elementos.
- ↗ Escreva uma função que contabiliza a quantidade de vezes que cada letra aparece no array de nomes.