

O que são ARRAYS?

Arrays são coleções **ordenadas** de valores. Cada valor é chamado **elemento** e tem sua posição na lista determinada por um **índice**.

const arr = [**0** elemento , **1** elemento , **2** elemento , **3** elemento , **4** elemento];

O primeiro índice em um array é sempre o número **zero**.

Valores são inseridos entre **colchetes []**, separados por **vírgula**,

Arrays em JS são **dinâmicas** e podem aumentar ou diminuir em quantidade de elementos conforme necessário, de **[]** a **[+4bi]**

Sintaxe do Array

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

Arrays podem
conter qualquer
tipo de dado em JS

Sintaxe do Array

```
const arrNumeros = [1, 2, 3];  
  
const arrMisto = [5, true, 'texto'];
```

Em JS um array pode
conter **tipos de dados
diferentes**

Sintaxe do Array

```
const arrNumeros = [1, 2, 3];  
  
const arrMisto = [5, true, 'texto'];  
  
const arrArrays = [[1, 2], ['a', 'b', 'c'], [true, false]];
```

É possível construir um **array de arrays** com tipos de dados e tamanhos diferentes. Cada array interno é um único elemento e é acessado através de um **índice**.

Sintaxe do Array

```
const arrNumeros = [1, 2, 3];  
  
const arrMisto = [5, true, 'texto'];  
  
const arrArrays = [[1, 2], ['a', 'b', 'c'], [true, false]];  
  
const arrObjetos = [{ a: 1 }, { b: 2 }, { c: 3 }];
```

Uma estrutura comum é o **array de objetos**. Cada objeto é um único **elemento** e é acessado através de um **índice**.

Manipulando Arrays

```
const arrNumeros = [46, 34, 23, 12];  
  
console.log(arrNumeros[0]);  
console.log(arrNumeros[1]);  
console.log(arrNumeros[2]);  
console.log(arrNumeros[3]);
```

O **acesso aos elementos** é sempre feito através do **índice correspondente**

Usamos a sintaxe
nomeVariavel[i]

Manipulando Arrays

```

| | | | | // 0 1 2 3
const arrNumeros = [46, 34, 23, 12];

arrNumeros[1] = 36;
arrNumeros[4] = 100;

console.log(arrNumeros);
//[ 46, 36, 23, 12, 100 ]
```

É possível **modificar valores** em um array a partir do **índice**, usando o operador de atribuição =

É possível **adicionar novos valores** a partir do **índice**, usando o operador de atribuição =

Acessando e manipulando arrays com FOR

```
const arrNumeros = [46, 34, 23, 12];  
  
for (let i = 0; i < arrNumeros.length; i++) {  
  console.log(arrNumeros[i]);  
}
```

O **for** utiliza o valor da variável **i** como posição do índice do array

arrNumeros.length → 4
índices → 0 a 3

Usamos a sintaxe **nomeVariavel[i]** para acessar um **elemento** a cada iteração

Acessando e manipulando arrays com FOR

```
const arrNumeros = [46, 34, 23, 12];  
for (let i = 0; i < arrNumeros.length; i++) {  
  arrNumeros[i] = arrNumeros[i] * 10;  
}  
  
//[ 460, 340, 230, 120 ]  
console.log(arrNumeros);
```

É possível manipular os valores usando a sintaxe **variavel[i]** e operadores

Acessando e manipulando arrays com FOR

```
const estudantes = ["JULiana", "aline", "SOLANGE"];

for (let i = 0; i < estudantes.length; i++) {
  estudantes[i] = estudantes[i].toUpperCase();
}

// [ 'JULIANA', 'ALINE', 'SOLANGE' ]
console.log(estudantes);
```

É possível manipular os valores usando a sintaxe **variavel[i]** com métodos e funções

Acessando e manipulando arrays com FOR...OF

```
const arrayNumeros = [18, 95, 45, 76, 23, 99];
```

```
for (let numero of arrayNumeros) {  
  if (numero + 10 > 100 || numero > 100) continue;  
  console.log(numero + 10);  
};
```

Pode ser usado com qualquer **iterável** (array, string, map, set, etc).

Laço pode ser manipulado com instruções como **break** e **continue**.

Métodos de Array

```
const arrNumeros = [12, 23, 34, 45, 56];
```

```
arrNumeros.push(67);  
console.log(arrNumeros);  
//[ 12, 23, 34, 45, 56, 67 ]
```

o método
array.push(<valor>)
insere o valor na **última**
posição do array

[métodos de Array no MDN](#)

Métodos de Array

```
const arrNumeros = [12, 23, 34, 45, 56];
```

```
arrNumeros.push(67);  
console.log(arrNumeros);  
//[ 12, 23, 34, 45, 56, 67 ]
```

```
arrNumeros.pop();  
console.log(arrNumeros);  
//[ 12, 23, 34, 45, 56 ]
```

o método **array.pop()**
remove o valor da **última**
posição do array

Métodos de Array

```
const arrNumeros = [12, 23, 34, 45, 56];

arrNumeros.push(67);
console.log(arrNumeros);
//[ 12, 23, 34, 45, 56, 67 ]

arrNumeros.pop();
console.log(arrNumeros);
//[ 12, 23, 34, 45, 56 ]

const elem = arrNumeros.indexOf(12);
console.log(elem);
//0
```



o método **array.indexOf(<valor>)**
localiza o valor do parâmetro e
retorna a posição do índice ou **-1**
caso não exista

Métodos de Array

```
const arrNumeros = [12, 23, 34, 45, 56];
```

```
arrNumeros.push(67);  
console.log(arrNumeros);  
//[ 12, 23, 34, 45, 56, 67 ]
```

```
arrNumeros.pop();  
console.log(arrNumeros);  
//[ 12, 23, 34, 45, 56 ]
```

```
const elem = arrNumeros.indexOf(12);  
console.log(elem);  
//0
```

```
const novoArr = arrNumeros.slice(2);  
console.log(novoArr);  
//[ 34, 45, 56 ]
```

o método **array.slice(<índice>)**
copia os valores a partir do índice e
cria um novo array mantendo o
anterior inalterado

Métodos de Array - callbacks

```
const arrNumeros = [12, 23, 34, 45, 56];

const arrCalculado = arrNumeros.map((num) => {
  return num * 10;
});

//[ 120, 230, 340, 450, 560 ]
console.log(arrCalculado);
```

o método **array.map(callback)** executa o código na função callback para cada elemento e **retorna o resultado** para um novo array

Métodos de Array - callbacks

```
const arrNumeros = [12, 23, 34, 45, 56];


const arrCalculado = arrNumeros.forEach((num, i) => {
  console.log(`o número ${num} está no índice ${i}`)
});
```

```
// o número 12 está no índice 0
// o número 23 está no índice 1
// o número 34 está no índice 2
// o número 45 está no índice 3
// o número 56 está no índice 4
```

o método **array.forEach(callback)** apenas executa o código na função callback para cada elemento. Todos os métodos callback têm como **segundo parâmetro o índice** do array.

Métodos de Array - callbacks

```
const arrNumeros = [12, 23, 34, 45, 56];  
  
const arrFiltrado = arrNumeros.filter(num => num % 5 === 0);  
  
//[ 45 ]  
console.log(arrFiltrado);
```



o método **array.filter(callback)** filtra elementos do array a partir de operações que resultem em **true/false**. Elementos que retornam **true** são retornados para um novo array.

Compartilhe um resumo de seus novos
conhecimentos em suas redes sociais.

[#aprendizadoalura](#)

alura



Escola Programação