

JavaScript

<Arrays>

Linguagens Script @ LEI / LEI-PL / LEI-CE

Departamento de Engenharia Informática e de Sistemas

Cristiana Areias < cris@isec.pt >

2023/2024

Javascript

Arrays

- › Características dos Arrays
 - › Declaração de Arrays
 - › Acesso aos elementos
 - › Métodos
- › Desestruturando (*Destructuring*) Arrays

> Arrays

- Um *array* é uma **estrutura de dados** que segue uma sequência não ordenada.

```
let array =[];
```

- Permite armazenar uma lista de itens dentro deles, e pode ser de qualquer tipo ou formato: Lista de produtos , lista de alunos, lista de disciplinas, lista de cores, ...

```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria', 'João'];
```



> Arrays > Características

- Redimensionáveis e podem conter diferentes tipos de dados**
 - Quando não se pretendem estas características podem-se usar *arrays* tipados (*ArrayBuffer* e *DataView*);
- Não são arrays associativos***, isto é, não podem ser acedidos usando *strings* como índices, mas devem ser acedidos usando números inteiros como índices
 - Em termos históricos, objetos simples eram usados para criar *arrays* associativos, pois tecnicamente fazem o mesmo. Atualmente, o objeto **Map** permite criar objetos associativos mas não é compatível com browser antigos.
- Indexados a 0**, o primeiro elemento do *array* inicia no índice 0 e o segundo no 1. O último elemento é o comprimento do array, menos 1.
 - O comprimento do array está especificado na propriedade *length*
- A copia de um array, é uma copia superficial.**

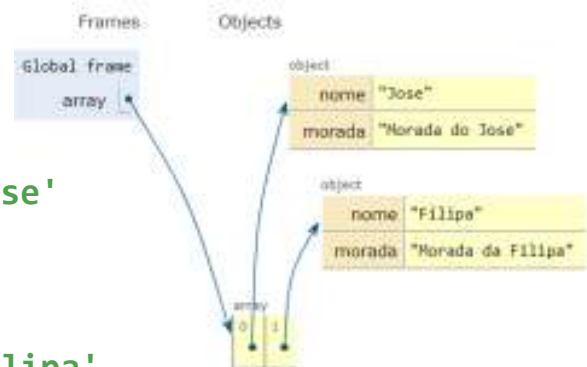
> Arrays > Vários tipos (prop.)

```
const alunoNuno = ['Nuno Afonso',  
                  2102124,  
                  'Rua Nova, Coimbra',  
                  ['AP', 'LS', 'TW', 'SO', 'AM', 'TAC']  
                  ];  
  
console.log(alunoNuno);
```

```
▼ (4) ['Nuno Afonso', 2102124, 'Rua Nova, Coimbra', Array(6)] ⓘ  
  0: "Nuno Afonso"  
  1: 2102124  
  2: "Rua Nova, Coimbra"  
  3: Array(6)  
    0: "AP"  
    1: "LS"  
    2: "TW"  
    3: "SO"  
    4: "AM"  
    5: "TAC"  
    length: 6  
    ► [[Prototype]]: Array(0)  
    length: 4  
    ► [[Prototype]]: Array(0)
```

> Arrays de Objetos

```
let array = [  
  {  
    nome: 'Jose',  
    morada: 'Morada do Jose'  
  },  
  {  
    nome: 'Filipa',  
    morada: 'Morada da Filipa'  
  }  
];
```



```
▼ (2) [{...}, {...}] ⓘ  
  ► 0: {nome: 'Jose', morada: 'Morada do Jose'}  
  ► 1: {nome: 'Filipa', morada: 'Morada da Filipa'}  
    length: 2  
  ► [[Prototype]]: Array(0)
```

> Arrays > Declaração com **const**?

- É possível alterar a variável `alunos` estando declarada com **const**?



```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria', 'João'];  
console.log(alunos);  
alunos[2] = 'Manuel';  
console.log(alunos);
```

```
▶ (6) ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria', 'João']  
▶ (6) ['Nuno', 'Ricardo', 'Manuel', 'José', 'Maria', 'João']
```

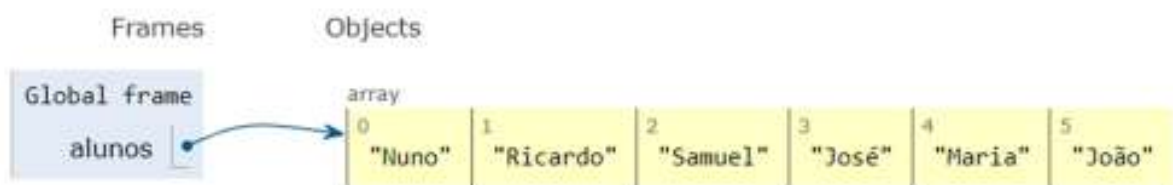
- Todos os tipos primitivos definem valores **imutáveis**
 - O que não é o caso dos arrays!

```
alunos = ['Filipa', 'Ana'];
```

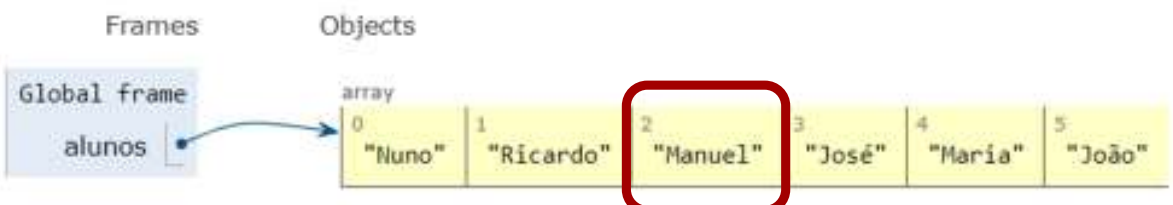


> Arrays > Declaração com **const**?

```
const alunos =  
  ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria', 'João'];  
console.log(alunos);
```



```
alunos[2] = 'Manuel';  
console.log(alunos);
```



> Arrays > Formas de Declaração

▪ Array Literal

- Simplicidade, legibilidade e velocidade de execução

```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria', 'João'];  
console.log(alunos);
```

► (6) ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria', 'João']

```
var Array: ArrayConstructor  
new <string>(...items: string[]) => string[] (+2 overloads)  
  
const alunosV2 = new Array('Nuno', 'Ricardo', 'Samuel', 'José', 'Maria', 'João');  
console.log(alunosV2);
```

Array constructor

A ver mais tarde...

> Array > Typeof

- Para identificar o tipo de uma variável recorre-se ao **typeof**, no entanto, no caso do Array, será retornado o valor “object”.
- Para saber se o element é de facto um array, pode-se usar o método
 - `Array.isArray (nomeArray)` - Retorna true ou false

```
const alunos = ['Nuno', 'Ricardo', 'Samuel'];  
console.log(typeof alunos);  
console.log(Array.isArray(alunos))
```

object
true



> Arrays > Acesso aos items

- Para aceder a elementos de um array, podemos recorrer ao seu índice.

```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria', 'João'];
```



```
console.log(alunos[0]);  
console.log(alunos[1]);  
  
console.log(alunos.length);
```

Nuno
Ricardo
6

propriedade

```
console.log(alunos);  
alunos[2]='Manuel';  
console.log(alunos);
```

▶ (6) ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria', 'João']
▶ (6) ['Nuno', 'Ricardo', 'Manuel', 'José', 'Maria', 'João']

> Arrays > Propriedades

- JavaScript inclui um três propriedades que podem ser aplicadas directamente a *arrays*.

Property	Description
<u>constructor</u>	Returns the function that created the Array object's prototype
<u>length</u>	Sets or returns the number of elements in an array
<u>prototype</u>	Allows you to add properties and methods to an Array object

https://www.w3schools.com/jsref/jsref_obj_array.asp

```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José'];  
console.log(alunos.length); //4  
console.table(alunos);  
alunos.length = 2;  
console.table(alunos);
```

(index)	Value
0	'Nuno'
1	'Ricardo'
2	'Samuel'
3	'José'

(index)	Value
0	'Nuno'
1	'Ricardo'

> Arrays > Métodos

- JavaScript inclui *built-in functions* que podem ser aplicadas diretamente aos *arrays*, que se designam como **métodos**
 - Permitem efetuar determinadas operações, como inserir novos elementos ao array, eliminar, ordenar. Exemplos:
 - `push()` / `pop()`
 - `shift()` / `unshift()`
 - `concat()`
 - `slice()`
 - `splice()`
 - `map()`
 - `reduce()` ...
 - Lista completa dos métodos e respetivas descrições disponível em
 - https://www.w3schools.com/jsref/jsref_obj_array.asp

> Arrays > Métodos

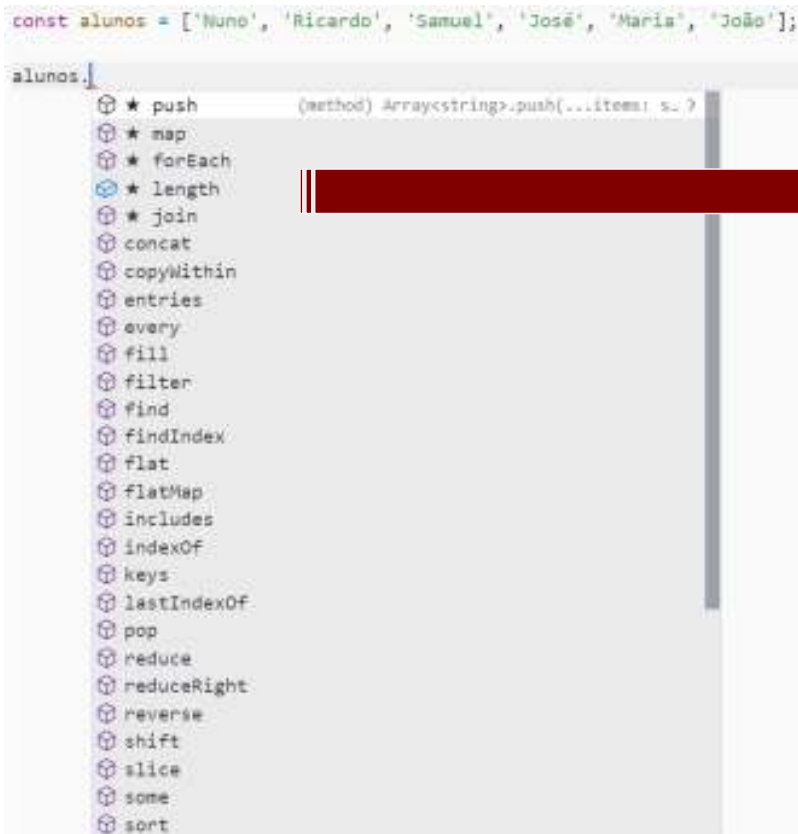
Method	Description
<code>concat()</code>	Joins two or more arrays, and returns a copy of the joined arrays
<code>copyWithin()</code>	Copies array elements within the array, to and from specified positions
<code>entries()</code>	Returns a key/value pair Array Iteration Object
<code>every()</code>	Checks if every element in an array pass a test
<code>fill()</code>	Fill the elements in an array with a static value
<code>filter()</code>	Creates a new array with every element in an array that pass a test
<code>find()</code>	Returns the value of the first element in an array that pass a test
<code>findIndex()</code>	Returns the index of the first element in an array that pass a test
<code>forEach()</code>	Calls a function for each array element
<code>from()</code>	Creates an array from an object
<code>includes()</code>	Check if an array contains the specified element
<code>indexOf()</code>	Search the array for an element and returns its position
<code>isArray()</code>	Checks whether an object is an array
<code>join()</code>	Joins all elements of an array into a string
<code>keys()</code>	Returns a Array Iteration Object, containing the keys of the original array
<code>lastIndexOf()</code>	Search the array for an element, starting at the end, and returns its position

Method	Description
<code>map()</code>	Creates a new array with the result of calling a function for each array element
<code>pop()</code>	Removes the last element of an array, and returns that element
<code>push()</code>	Adds new elements to the end of an array, and returns the new length
<code>reduce()</code>	Reduce the values of an array to a single value (going left-to-right)
<code>reduceRight()</code>	Reduce the values of an array to a single value (going right-to-left)
<code>reverse()</code>	Reverses the order of the elements in an array
<code>shift()</code>	Removes the first element of an array, and returns that element
<code>slice()</code>	Selects a part of an array, and returns the new array
<code>some()</code>	Checks if any of the elements in an array pass a test
<code>sort()</code>	Sorts the elements of an array
<code>splice()</code>	Adds/Removes elements from an array
<code>toString()</code>	Converts an array to a string, and returns the result
<code>unshift()</code>	Adds new elements to the beginning of an array, and returns the new length
<code>valueOf()</code>	Returns the primitive value of an array

> Arrays > Métodos

```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria', 'João'];

alunos.
```



> Arrays > Métodos > *push*

```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria'];
```

```
▼ (5) ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria'] ⓘ  
  0: "Nuno"  
  1: "Ricardo"  
  2: "Samuel"  
  3: "José"  
  4: "Maria"  
  length: 5  
  ► [[Prototype]]: Array(0)
```

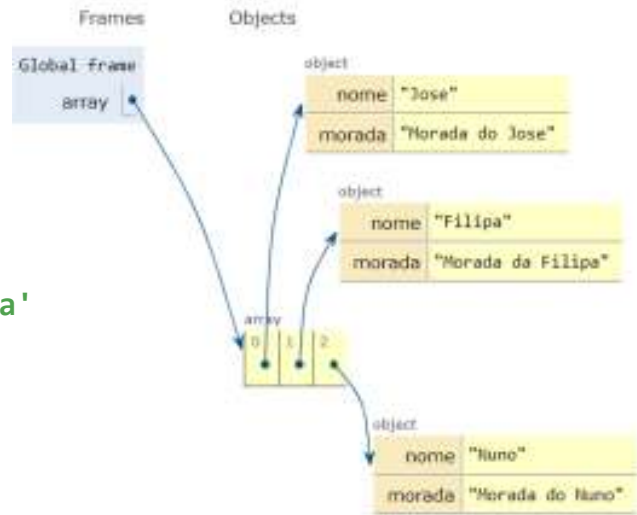
```
alunos.push("João")  
alunos[alunos.length] = "OutroAluno";
```

```
▼ (7) ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria', 'João', 'OutroAluno'] ⓘ  
  0: "Nuno"  
  1: "Ricardo"  
  2: "Samuel"  
  3: "José"  
  4: "Maria"  
  5: "João"  
  6: "OutroAluno"  
  length: 7  
  ► [[Prototype]]: Array(0)
```


> Arrays > Métodos > *push*

```
let array = [  
  {  
    nome: 'Jose',  
    morada: 'Morada do Jose'  
  },  
  {  
    nome: 'Filipa',  
    morada: 'Morada da Filipa'  
  }  
];
```

```
array.push({  
  nome: 'Nuno',  
  morada: 'Morada do Nuno'  
});
```



```
▼ (3) [(-), {(-)}, {(-)}] 0  
  ▶ 0: {nome: 'Jose', morada: 'Morada do Jose'}  
  ▶ 1: {nome: 'Filipa', morada: 'Morada da Filipa'}  
  ▶ 2: {nome: 'Nuno', morada: 'Morada do Nuno'}  
    length: 3  
  ▶ [[Prototype]]: Array(0)
```

> Arrays > Métodos > *unshift*

```
(method) Array<string>.unshift(...items: string[]): number  
Inserts new elements at the start of an array, and returns the new length of  
the array.  
@param items — Elements to insert at the start of the array.
```

```
alunos.unshift("Carlos");  
console.log(alunos);
```

```
▶ (7) ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria', 'João', 'Afonso'] scrip  
▼ (8) ['Carlos', 'Nuno', 'Ricardo', 'Samuel', 'José', 'Maria', 'João', 'Afonso'] scrip  
  0: "Carlos"  
  1: "Nuno"  
  2: "Ricardo"  
  3: "Samuel"  
  4: "José"  
  5: "Maria"  
  6: "João"  
  7: "Afonso"  
    length: 8  
  ▶ [[Prototype]]: Array(0)
```

> Arrays > Métodos > *shift*

(method) `Array<string>.pop(): string`

Removes the last element from an array and returns it. If the array is empty, undefined is returned and the array is not modified.

```
alunos.pop();
alunos.pop();
const alunoEliminado = alunos.pop();
console.log(alunos);
console.log(alunoEliminado);
```

```
► (8) ['Carlos', 'Nuno', 'Ricardo', 'Samuel', 'José', 'Maria', 'João', 'Afonso']
► (5) ['Carlos', 'Nuno', 'Ricardo', 'Samuel', 'José']
Maria
```

```
alunos.shift();
console.log(alunos);
```

```
► (4) ['Nuno', 'Ricardo', 'Samuel', 'José']
```

> Arrays > Métodos > *push*

```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria'];
```

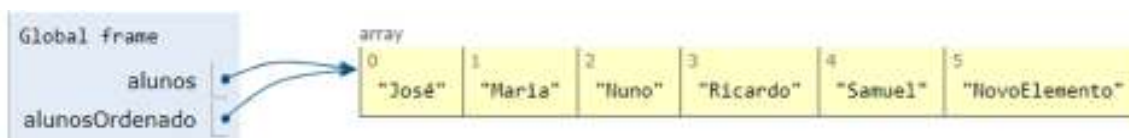
```
let alunosOrdenado = alunos.sort();
alunosOrdenado.push("NovoElemento");
console.table(alunos);
console.table(alunosOrdenado);
```

alunos

(index)	Value
0	'José'
1	'Maria'
2	'Nuno'
3	'Ricardo'
4	'Samuel'
5	'NovoElemento'

alunosOrdenado

(index)	Value
0	'José'
1	'Maria'
2	'Nuno'
3	'Ricardo'
4	'Samuel'
5	'NovoElemento'



> Arrays > Métodos > *pop* e *shift*

```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria'];
```

```
alunos.pop();  
alunos.shift();  
alunos.concat(['novo1', 'novo2']);  
let alunosNovo =  
alunos.concat(['novo1', 'novo2']);
```

alunos

(index)	Value
0	'Ricardo'
1	'Samuel'
2	'José'

alunosNovo

(index)	Value
0	'Ricardo'
1	'Samuel'
2	'José'
3	'novo1'
4	'novo2'

> Arrays > Métodos > *splice*

```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria'];
```

```
let alunosNovo = alunos.splice(1, 1, 'Filomena');
```

(method) `Array<string>.splice(start: number, deleteCount: number, ...items: string[])`:
`string[]` (+1 overload)

Removes elements from an array and, if necessary, inserts new elements in their place, returning the deleted elements.

@param start — The zero-based location in the array from which to start removing elements.

@param deleteCount — The number of elements to remove.

@param items — Elements to insert into the array in place of the deleted elements.

@returns — An array containing the elements that were deleted.

```
splice(1, 1, 'Filomena');
```

> Arrays > Métodos > *splice*

```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria'];
```

```
let alunosNovo = alunos.splice(1, 1, 'Filomena');
```

alunos

(index)	Value
0	'Nuno'
1	'Filomena'
2	'Samuel'
3	'José'
4	'Maria'

alunosNovo

(index)	Value
0	'Ricardo'

> Arrays > Métodos > *push*

```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria'];
```

```
let alunosNovo = alunos.slice(0, 4);
```

(method) `Array<string>.slice(start?: number, end?: number): string[]`

Returns a copy of a section of an array. For both start and end, a negative index can be used to indicate an offset from the end of the array. For example, -2 refers to the second to last element of the array.

@param start

The beginning index of the specified portion of the array. If start is undefined, then the slice begins at index 0.

@param end

The end index of the specified portion of the array. This is exclusive of the element at the index 'end'. If end is undefined, then the slice extends to the end of the array.

```
is.slice(0, 4);
```

> Arrays > Métodos > *slice*

```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria'];
```

```
let alunosNovo = alunos.slice(0, 4);  
let alunosNovo2 = alunos.slice(3, 4);
```

alunos

(index)	Value
0	'Nuno'
1	'Ricardo'
2	'Samuel'
3	'José'
4	'Maria'

alunosNovo

(index)	Value
0	'Nuno'
1	'Ricardo'
2	'Samuel'
3	'José'

alunosNovo2

(index)	Value
0	'José'

> Arrays > Métodos > *push*

```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria'];
```

```
let alunosNovo = alunos.slice();  
alunosNovo.pop();  
alunosNovo.pop();  
alunos.push("ola");
```



alunos

(index)	Value
0	'Nuno'
1	'Ricardo'
2	'Samuel'
3	'José'
4	'Maria'
5	'ola'

alunosNovo

(index)	Value
0	'Nuno'
1	'Ricardo'
2	'Samuel'

> Arrays > Métodos > *join*

```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria'];
```

```
let alunosNovo = alunos.join(',')
```

alunos

(index)	Value
0	'Nuno'
1	'Ricardo'
2	'Samuel'
3	'José'
4	'Maria'

alunosNovo

Nuno,Ricardo,Samuel,José,Maria



string

> Arrays > Métodos > *sort*

```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria'];
```

```
alunos.sort();
```

(index)	Value
0	'José'
1	'Maria'
2	'Nuno'
3	'Ricardo'
4	'Samuel'

```
const notas = [13,9,10,19,7,20];
```

```
alunos.sort();
```

(index)	Value
0	10
1	13
2	19
3	20
4	7
5	9



> Arrays > Métodos > *sort*

```
const notas = [13, 9, 10, 19, 7, 20];
```

```
notas.sort(function (a, b) {  
    return a - b;  
});
```



```
notas.sort((a, b) => a - b);
```



Função de Comparação

Baseado na diferença de dois valores
 $7 - 9 = \text{negativo}$

...

(index)	Value
0	7
1	9
2	10
3	13
4	19
5	20

> Arrays > Métodos > *map*

```
let arrayNumeros = [5, 6, 7, 8];
```

(index)	Value
0	5
1	6
2	7
3	8



(index)	Value
0	15
1	18
2	21
3	24

```
for (let i = 0; i < arrayNumeros.length; i++) {  
    arrayNumeros[i] = arrayNumeros[i] * 3;  
}
```


> Arrays > Métodos > *map*

```
let arrayNumeros = [5, 6, 7, 8];
```

```
arrayNumeros=arrayNumeros.map(function (numero) {  
    return numero * 3;  
});
```



```
arrayNumeros = arrayNumeros.map(num => num * 3);
```

> Arrays > Métodos > *map*

```
let arrayNumeros = [5, 6, 7, 8, 6 ,7];
```

```
let novo = arrayNumeros.map(function (numero) {  
    if (numero != arrayNumeros.length)  
        return numero * 3;  
});
```

```
console.log(novo);  
console.log(arrayNumeros);
```

(índice)	Valor
0	15
1	undefined
2	21
3	24
4	undefined
5	21

(índice)	Valor
0	5
1	6
2	7
3	8
4	6
5	7

> Arrays > Métodos > *map*

```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria'];
```

(index)	Value
0	'Nuno-LS'
1	'Ricardo-LS'
2	'Samuel-LS'
3	'José-LS'
4	'Maria-LS'



```
let alunosLS = alunos.map(alteraNomeAluno);
```

```
function alteraNomeAluno(aluno) {  
    return `${aluno}-LS`;  
}
```

> Arrays > Métodos > *map*

```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria'];
```

```
let alunosLS = alunos.map(function (aluno) {  
    return `${aluno}-LS`;  
});
```

```
let alunosLS = alunos.map(aluno => `${aluno}-LS`);
```

```
▼ (5) ['Nuno-LS', 'Ricardo-LS', 'Samuel-LS', 'José-LS', 'Maria-LS']  
  0: "Nuno-LS"  
  1: "Ricardo-LS"  
  2: "Samuel-LS"  
  3: "José-LS"  
  4: "Maria-LS"  
  length: 5  
  ► [[Prototype]]: Array(0)
```

Mais exemplos no contexto
das funções



> Arrays > Métodos > *filter*

```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria'];
```

(index)	Value
0	'Samuel'
1	'Maria'

```
▼ (2) ['Samuel', 'Maria'] ⓘ  
  0: "Samuel"  
  1: "Maria"  
  length: 2  
  ► [[Prototype]]: Array(0)
```



```
let alunosLS = alunos.filter(function (aluno) {  
  return aluno[1] === 'a';  
});
```

Mais exemplos no contexto
das funções



> Arrays > Métodos > *reduce*

```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria'];
```

```
let alunosLS = alunos.reduce(function (a1, a2) {  
  return `${a1},${a2}`;  
});
```

Nuno,Ricardo,Samuel,José,Maria

```
const notas = [13,9,10,19,7,20];
```

```
let soma = notas.reduce(function (n1, n2) {  
  return n1 + n2;  
});  
console.log(soma);  
console.log(soma / notas.length);
```

78

13

> Arrays > Outros Métodos

```
console.log(alunos);
console.log(alunos.indexOf('José'));
console.log(alunos.indexOf('Joseeee'));
console.log(alunos.includes('José'));
console.log(alunos.includes('JOSÉ'));
```

```
► (4) ['Nuno', 'Ricardo', 'Samuel', 'José']
3
-1
true
false
```

```
const nome='José'
if (alunos.includes(nome))
|   console.log(`O ${nome} é aluno de Linguagens Script!`);
else
|   console.log(`O ${nome} é não aluno de Linguagens Script!`);
```

```
► (4) ['Nuno', 'Ricardo', 'Samuel', 'José']
O José é aluno de Linguagens Script!
```

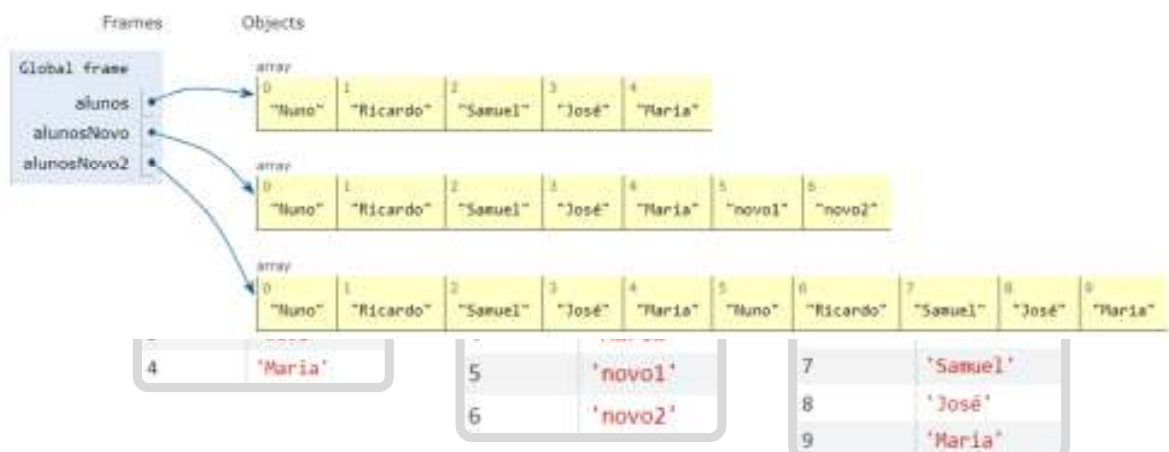
> Arrays > *spread operator*

```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria'];
```

```
let alunosNovo = [...alunos, "novo1", "novo2"]
let alunosNovo2 = [...alunos, ...alunos]
```



...
Spread
Operator



> Arrays > *spread Operator*

```
const alunos = ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria'];
```

```
let alunosNovo = alunos.push(...alunos);
```

alunos

(index)	Value
0	'Nuno'
1	'Ricardo'
2	'Samuel'
3	'José'
4	'Maria'
5	'Nuno'
6	'Ricardo'
7	'Samuel'
8	'José'
9	'Maria'

alunosNovo

10

> Arrays > *spread Operator*

```
let palavras = ['pa1', 'pa2', 'pa3', 'pa4'];
```

```
let oQueSera = [...palavras[0]];
```

```
(3) ['p', 'a', '1']  
  0: "p"  
  1: "a"  
  2: "1"  
length: 3
```

> *Destructuring*

- ES6 introduziu uma característica sintática designada como *destructuring*
- A sintaxe de atribuição via desestruturação (*destructuring assignment*) é uma expressão JavaScript que permite extrair dados de um **Array** ou de um **Objecto** para variáveis distintas.

```
const disc = ["Algoritmos", "Linguagens Script",  
              "Tecnologias Web", "Programação"];  
  
const ap = disc[0];  
const ls = disc[1];  
const tw = disc[2];
```

Pré-ES2015



> *Destructuring Arrays*

```
const disc = ["Algoritmos", "Linguagens Script", "Tecnologias Web",  
              "Programação"];  
  
const [ap, ls, tw, p, tac] = disc;  
  
console.log(disc);  
console.log(ap);  
console.log(ls);  
console.log(tw);  
console.log(p);  
console.log(tac);
```



Pós-ES2015

```
► (4) ['Algoritmos', 'Linguagens Script', 'Tecnologias Web', 'Programação']  
Algoritmos  
Linguagens Script  
Tecnologias Web  
Programação  
undefined
```

> Destructuring Arrays

```
const disc = ["Algoritmos", "Linguagens Script", "Tecnologias Web",  
              "Programação"];  
const [ap, ls, tw, p, tac] = disc;  
console.log(disc);  
console.log(ap);  
console.log(ls);  
console.log(tw);  
console.log(p);  
console.log(tac);
```



Pós-ES2015

```
▶ (4) ['Algoritmos', 'Linguagens Script', 'Tecnologias Web', 'Programação']  
Algoritmos  
Linguagens Script  
Tecnologias Web  
Programação  
undefined
```

> Destructuring Arrays

```
const disc = ["Algoritmos", "Linguagens Script", "Tecnologias Web",  
              "Programação"];  
const [ap, ls, ...resto] = disc;  
console.log(ap);  
console.log(ls);  
console.log(resto);
```



Pós-ES2015

```
Algoritmos  
Linguagens Script  
▼ (2) ['Tecnologias Web', 'Programação'] ⓘ  
  0: "Tecnologias Web"  
  1: "Programação"  
  length: 2
```


> Destructuring Arrays

- Trocar valores de variáveis?

```
let nome1="Nuno Afonso";  
let nome2="Ricardo Afonso";  
console.log(nome1, " - ", nome2);
```

Nuno Afonso - Ricardo Afonso

```
[nome1,nome2] = [nome2,nome1];  
console.log(nome1 + " - " + nome2);
```

Nuno Afonso - Ricardo Afonso

Ricardo Afonso - Nuno Afonso

>



> Percorrer um array

- Muitas vezes, ao trabalhar com arrays precisamos fazer iterações sobre os arrays. Existem várias formas, entre elas:

- for
- for...of
- for...in
- foreach

**Imensas
formas!**



```
let frutas = ["Banana", "Laranja", "Maça", "Pera"];
```

```
for (let i = 0; i < frutas.length; i++) {  
    // Faz qualquer coisa...  
    console.log(frutas[i]);  
}
```

> Array > for...of

■ for...of

```
for (variavel of iteravel) {  
  // Código  
}
```

```
let frutas = ["Banana", "Laranja", "Maça", "Pera"];
```

```
for (const fruta of frutas) {  
  console.log(fruta);  
}
```

Banana
Laranja
Maça
Pera

```
let fruta = frutas[0];  
for (const valor of fruta) {  
  console.log(valor);  
}
```

B
a
n
a
n
a

> Array > for...in

■ for...in

```
for (variavel in iteravel) {  
  // Código  
}
```

```
let frutas = ["Banana", "Laranja", "Maça", "Pera"];
```

```
for (const f in frutas) {  
  console.log(f);  
}
```

0
1
2
3

```
for (const f in frutas) {  
  console.log(f+'-' + frutas[f]);  
}
```

0-Banana
1-Laranja
2-Maça
3-Pera

> Array > forEach

- O `forEach` executa o **callback** fornecido, uma vez para cada elemento da ordem com um valor atribuído. Isto é, a função **forEach** percorre o array passando por cada item.

```
arr.forEach(callback(currentValue [, index [, array]])[, thisArg]);
```

- `callback` é invocado com três argumentos:
 - o valor do elemento
 - o índice do elemento
 - o array que está sendo percorrido

```
frutas.forEach(function (fruta) {  
    // Faz qualquer coisa..  
});
```

Para cada fruta,
executar a função

> Array > forEach

```
frutas.forEach(function (fruta, indice, arrayFrutas) {  
    console.log(`A ${fruta} está na posicao${indice}  
                do array [${arrayFrutas}]`);  
});
```

```
A Banana está na posicao 0 do array [Banana,Laranja,Maça,Pera]  
A Laranja está na posicao 1 do array [Banana,Laranja,Maça,Pera]  
A Maça está na posicao 2 do array [Banana,Laranja,Maça,Pera]  
A Pera está na posicao 3 do array [Banana,Laranja,Maça,Pera]
```

> Array > forEach

- Simplificando...

```
let frutas = ["Banana", "Laranja", "Maça", "Pera"];  
console.log(frutas);
```

```
frutas.forEach((fruta, indice) =>  
  console.log(`[${indice}] = ${fruta}`)  
);
```

**Arrow
functions**



Secção funções em JavaScript



> Array de Objetos > for...in

```
const carro = {  
  marca: 'Toyota',  
  preco: 32000,  
  modelo: 'auries',  
  cor: 'Vermelho'  
}
```

```
for (const p in carro) {  
  console.log(p + '-' + carro[p]);  
}
```

marca-Toyota
preco-32000
modelo-auries
cor-Vermelho

> Array de Objetos > *map*

```
const carros = [
  { marca: 'Toyota', preco: 32000, modelo: 'auries', cor: 'Vermelho' },
  { marca: 'Renault', preco: 700, modelo: 'clio', color: 'Preto' },
  { marca: 'Audi', preco: 40000, modelo: 'A5', color: 'Cinza' },
];
```

```
const nomesCarros = carros.map(carro => carro.marca);
console.table(nomesCarros);
```

(index)	Value
0	'Toyota'
1	'Renault'
2	'Audi'

> Array de Objetos > *for...in*

```
const carros = [
  { marca: 'Toyota', preco: 32000, modelo: 'auries', cor: 'Vermelho' },
  { marca: 'Renault', preco: 700, modelo: 'clio', color: 'Preto' },
  { marca: 'Audi', preco: 40000, modelo: 'A5', color: 'Cinza' },
];
```

```
for (const carro in carros) {
  console.log(carro + '-' + carros[carro].marca);
}
```

```
0-Toyota
1-Renault
2-Audi
```

```
for (const p in carros[0]) {
  console.log(p + '-' + carros[0][p]);
}
```

```
marca-Toyota
preco-32000
modelo-auries
cor-Vermelho
```

</Arrays>

