JavaScript

<Tipos de Dados>

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

Departamento de Engenharia Informática e de Sistemas

Cristiana Areias < cris@isec.pt >

2022/2023

Tipos de Dados

- **Primitivos** - *Value Types*
  - *Boolean, String, Number,...*
- **Objetos** – *Reference Types*
  - *Objetos, Arrays, Funções*

Tipos de Dados

Primitivos	Objetos
Value Types	*Reference Types*

- Todo o valor é um valor **primitivo** ou um **objeto**.
  - Um **primitivo** é um dado que não é representado através de um objeto, por consequência, não possui métodos.
- Uma característica do JavaScript é **tipagem dinâmica**, logo:
  - não é necessário declarar o tipo da variável antes de ser atribuida;
  - Os tipos de dados são determinados de forma automática e podem alterar em tempo de execução;

```
let alunoAprovado = true;
console.log(typeof alunoAprovado);

alunoAprovado="José Meira";
console.log(typeof alunoAprovado);
```

```
boolean
string
```


Primitivos > *Value Types*

- A variável contém diretamente o **valor atribuído**;
- Todos os tipos primitivos são **imutáveis**;
- O mais recente padrão *ECMAScript* define sete tipos de dados primitivos;

Primitivos *Primitive Types*	
	Boolean
	Number
	String
	undefined
	null
ES2015	Symbol
ES2020	BigInt

```
let alunoAprovado = true;
let nota = 15;
let disciplina = "Linguagens Script";
let nomeAluno;
let alunoSeleccionado=null;
```

https://developer.mozilla.org

Primitivos > *Value Types*


```
> let disciplina = "Linguagens Script"; // String literal
  typeof disciplina;
< 'string'
> let nota = 15; // Number literal
  typeof nota;
< 'number'
> let alunoAprovado = true;
  typeof alunoAprovado;
< 'boolean'
> let nome;
  typeof nome;
< 'undefined'
> let nome=undefined;
  typeof nome;
< 'undefined'
> let nomeSeleccionado=null;
  typeof nomeSeleccionado;
< 'object'
```

undefined *– Indica variavel não inicializada*
´É um tipo mas também um valor

Indica um não valor, deliberadamente

objecto!

Strings > *Template Literals*

- Para a declaração de *strings* recorre-se às aspas simples ' ' ou duplas " ";
- O ES6 introduziu um novo tipo de "template literals" que recorre ao uso dos acentos graves ``
  - Permitem que expressões básicas de interpolação de *strings* sejam incorporadas, depois analisadas e avaliadas automaticamente;

Pré ES6

```
var ls = "Linguagens Script";
var info = "UC: " + ls + "!";
console.log(info);
console.log(typeof info);
```

ES6

```
var ls = "Linguagens Script";
var info = `UC: ${ls}!`;
console.log(info);
console.log(typeof info);
```

```
UC: Linguagens Script!
string
```

Strings > *Template Literals*

- Um dos benefícios da interpolação é o facto de permitirem especificar e dividir a *string* em várias linhas:

```
var ls = "Linguagens Script";
var sem = 2;
var info =
`UC: ${ls}!
      Disciplina do 1º Ano - ${sem} Semestre
      Licenciatura em Engenharia Informática e de Sistemas
ISEC`;
console.log(info)
```

```
UC: Linguagens Script!
      Disciplina do 1º Ano - 2 Semestre
      Licenciatura em Engenharia Informática e de Sistemas
ISEC
```


Primitivos – *Value Types*


```
let media = 15;
let mediaAntiga = media;
console.log(media);
console.log(mediaAntiga);
media = 12;
console.log(media);
console.log(mediaAntiga);
```

15
15
12
15

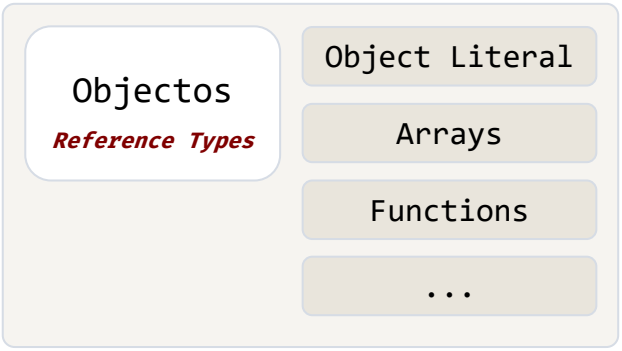
Localizações de memória diferentes!

Frames

Global frame

media	12
mediaAntiga	15

Objetos – *Reference Types*

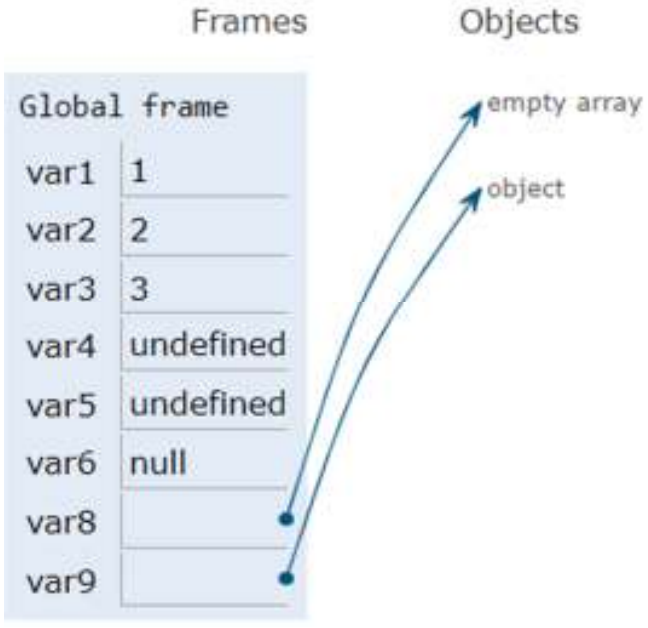



```
let varObj = {};
```

- Tudo o que não é primitivo, é **Objeto**!
- Objetos podem conter diferentes tipos de dados!
- Objetos, arrays, funções são tipos de dados **mutáveis;**
- Um objeto também pode ter funções (designados como métodos);
- Um objeto é composto por uma lista de pares entre propriedades e valores, associando uma chave com um valor;

Dados > valor vs referência


```
var var1 = 1;
let var2 = 2;
const var3 = 3;
let var4;
let var5 = undefined;
let var6 = null;
let var8 = [];
let var9 = {};
```



https://pythontutor.com/javascript.html

JavaScript

<Objetos>

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

Departamento de Engenharia Informática e de Sistemas

Cristiana Areias < cris@isec.pt >

2022/2023

Objetos

- O que são objetos?
- Objetos em JavaScript
  - Formas de Criação - Literal
  - Acesso e alteração das propriedades

O que é um Objeto?

- **Objetos** ➔ pensar em situações do mundo real…
  - **Algo visível**: um autocarro, chave, saco, livro…
  - **Algo que não se pode tocar**: tempo, um evento, uma conta bancária…

- **Objetos** podem ser vistos como um conjunto de **Propriedades**
  - **Características do objeto**: cor, nº de páginas num livro,..
  - Descrevem o estado corrente de um objeto.
  - O **estado** de um objeto é independente de outro. Exemplo: Um carro é amarelo e o outro é azul.

- O **Comportamento** refere-se ao que "permite fazer";
  - Numa conta bancária: um depósito, levantamento,…
- **Objetos são substantivos! Não são comportamentos nem propriedades!**

Objetos em JavaScript

- Objetos em JavaScript podem ser vistos como simples *coleções* compostas por pares **chave e valor**

```
const aluno = {
    nome: 'Manuel Afonso',
    numero: 1232123
}
```

Chave *Valor*

```
const aluno = { nome:'Manuel Afonso', numero:1232123 }
```

- Comparativamente com outras linguagens, podem ser vistos como:
  - *Hash tables* na linguagem C e C++
  - *HashMap* na linguagem Java
  - *Dicionários* na linguagem Python
  - …

Objetos > Criação

- Existem diferentes formas de se criar um **objeto**

Notação Literal `{}`

Instância de um Objeto `new Object()`

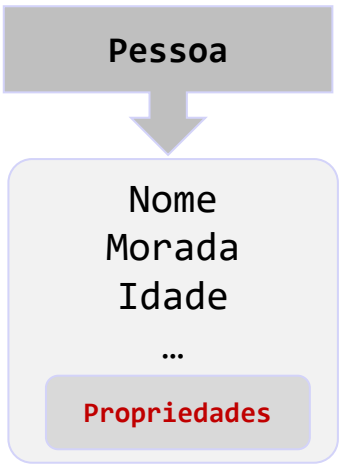
Definindo um Construtor `new Pessoa()`

Usando método Object.create `Object.create()`

- É muito comum criar um **objeto** usando um literal de objeto quando se pretende transferir um conjunto de dados estruturados relacionados.
- A criação de um **objeto literal** é tipicamente utilizada **para criar um único objeto**, enquanto que com recurso a um construtor é útil para criar múltiplos objetos.

Objetos > Criação


```
let nome;
let morada;
let idade;
```



Notação Literal

propriedades

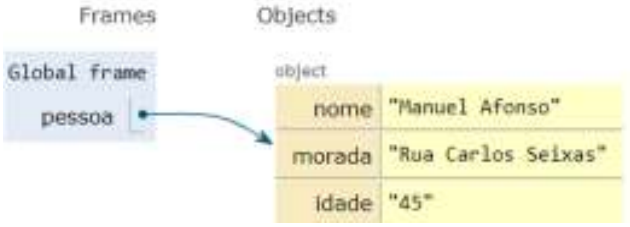
```
let pessoa = {
    nome:'Manuel Afonso',
    morada:'Rua Carlos Seixas',
    idade:'45'
}
```

Objetos > Criação

Notação Literal

```
let pessoa = {
    nome:'Manuel Afonso',
    morada:'Rua Carlos Seixas',
    idade:'45'
}
```

```
▼{nome: 'Manuel Afonso', morada: 'Rua Carlos Seixas',
idade: '45'} i
    idade: "45"
    morada: "Rua Carlos Seixas"
    nome: "Manuel Afonso"
  ▶ [[Prototype]]: Object
```



Objetos > Criação


```
const nuno = {
    nome:'Nuno Afonso',
    numero:2102124,
    morada:'Rua Nova, Coimbra',
    disciplinasInscritas:['AP','LS','TW','SO','AM','TAC']
}
```

Vários tipos!

```
▼{nome: 'Nuno Afonso', numero: 2102124, morada: 'Rua Nova, Coimbra', disciplinasInscritas: Array(6)}
  ▼disciplinasInscritas: Array(6)
      0: "AP"
      1: "LS"
      2: "TW"
      3: "SO"
      4: "AM"
      5: "TAC"
      length: 6
    ▶ [[Prototype]]: Array(0)
    morada: "Rua Nova, Coimbra"
    nome: "Nuno Afonso"
    numero: 2102124
  ▶ [[Prototype]]: Object
```

Objetos > Propriedades

- As propriedades podem ser acedidas ou alteradas de duas formas:

Dot notation

Notação com .

`objectName.objectProperty`

```
console.log(nuno.nome)
nuno.nome = "Nuno Afonso";
```

Bracket notation

Notação com [

`objectName['objectProperty']`

```
console.log(nuno['morada'])
nuno['morada'] = "Rua Velha";
```


Qual método usar?

› Dot Notation .

› Bracket notation []

Objetos > Acesso às Propriedades


```
const propriedade='morada';
console.log(nuno.propriedade);
console.log(nuno[propriedade]);
```

```
undefined
▶ ['Rua Velha, Coimbra']
```

```
console.log(`O aluno ${nuno.nome} com número ${nuno['numero']}
está inscrito a ${nuno.disciplinasInscritas.length} disciplinas:
${nuno.disciplinasInscritas}`);
```

```
O aluno Nuno Manuel Afonso com número 2102124
está inscrito a 6 disciplinas:
AP,LS,TW,SO,AM,TAC
```


Objetos > Acesso às Propriedades


```
const aluno1 = {
        nome: "Nuno Afonso",
        media: 15,
        cursoConcluido: false
};


const aluno2=aluno1;
aluno2.nome="Ricardo Afonso";

console.log(aluno1);
console.log(aluno2);
```

```
▶ {nome: 'Ricardo Afonso', media: 15, cursoConcluido: false}
▶ {nome: 'Ricardo Afonso', media: 15, cursoConcluido: false}
```

Objetos > Acesso às Propriedades


```
const aluno1 = {
        nome: "Nuno Afonso",
        media: 15,
        cursoConcluido: false
};


const aluno2=aluno1;
aluno2.nome="Ricardo Afonso";

aluno2 = {};
console.log(aluno2);
```

```
Uncaught TypeError: Assignment to constant variable.
```


Objetos > Alteração Propriedades


```
const aluno1 = {
        nome: "Nuno Afonso",
        media: 15,
        cursoConcluido: false
};

const aluno2 = Object.assign({},aluno1);
console.log(aluno1);
console.log(aluno2);

aluno2.nome="Ricardo Afonso";
console.log(aluno1);
console.log(aluno2);
```

Tem limitações!

```
▶ {nome: 'Nuno Afonso', media: 15, cursoConcluido: false}
▶ {nome: 'Ricardo Afonso', media: 15, cursoConcluido: false}
```

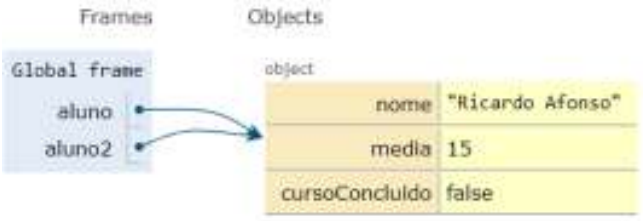
Objetos > Alteração Propriedades


```
const aluno = {
      nome: "Nuno Afonso",
      media: 15,
      cursoConcluido: false
};
```

```
let aluno2 = aluno;
console.log(aluno);
console.log(aluno2);

aluno2.nome = "Ricardo Afonso";

console.log(aluno);
console.log(aluno2);

aluno2.novaProp = "Prop. Nova";
console.log(aluno);
console.log(aluno2);

aluno2 = { nome: "Ana Afonso",
           idade: 30 }
console.log(aluno);
console.log(aluno2);
```


Objetos > Alteração Propriedades

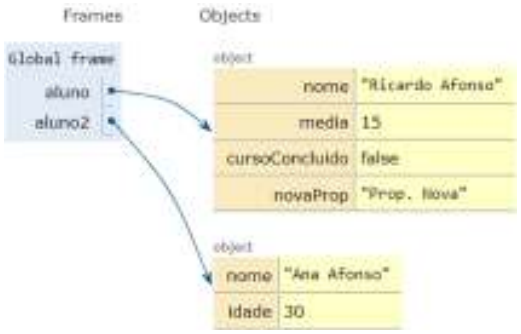

```
let aluno2 = aluno;
console.log(aluno);
console.log(aluno2);

aluno2.nome = "Ricardo Afonso";
console.log(aluno);
console.log(aluno2);
```




```
▶ {nome: 'Nuno Afonso', media: 15, cursoConcluido: false}
▶ {nome: 'Nuno Afonso', media: 15, cursoConcluido: false}
▶ {nome: 'Ricardo Afonso', media: 15, cursoConcluido: false}
▶ {nome: 'Ricardo Afonso', media: 15, cursoConcluido: false}
```

Objetos > Alteração Propriedades

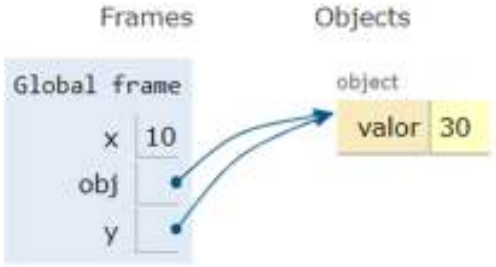

```
aluno2.novaProp = "Prop. Nova";
console.log(aluno);
console.log(aluno2);

aluno2 = {
          nome: "Ana Afonso",
          idade: 30 }

console.log(aluno);
console.log(aluno2);
```




```
▶ {nome: 'Ricardo Afonso', media: 15, cursoConcluido: false, novaProp: 'Prop. Nova'}
▶ {nome: 'Ricardo Afonso', media: 15, cursoConcluido: false, novaProp: 'Prop. Nova'}
▶ {nome: 'Ricardo Afonso', media: 15, cursoConcluido: false, novaProp: 'Prop. Nova'}
▶ {nome: 'Ana Afonso', idade: 30}
```


Objetos – Reference Types


```
let x = 10;
let obj = { valor: 20 };
let y = obj;
obj.valor = 30;

console.log(x);
console.log(y);
console.log(obj.valor);
console.log(y.valor);
```




```
10
▼ Object i
    valor: 30
  ▶ [[Prototype]]: Object
30
30
```

JavaScript

<Arrays>

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

Departamento de Engenharia Informática e de Sistemas

Cristiana Areias < cris@isec.pt >

2022/2023

Arrays

Arrays

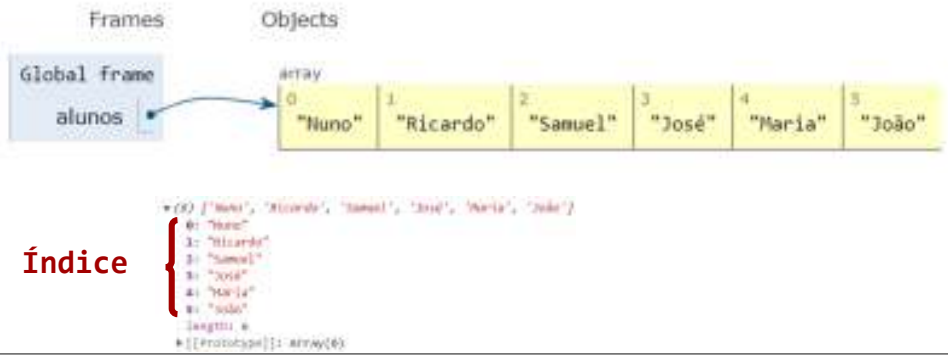
JavaScript

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

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

- Permitem 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

JavaScript

- **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.)

JavaScript

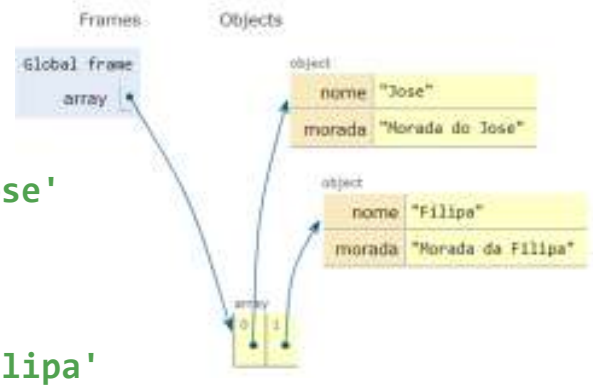
```
const alunoNuno = ['Nuno Afonso',
                   2102124,
                   'Rua Nova, Coimbra',
                   ['AP','LS','TW','SO','AM','TAC']
                  ];

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

JavaScript

```
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 **co**

```
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']
```

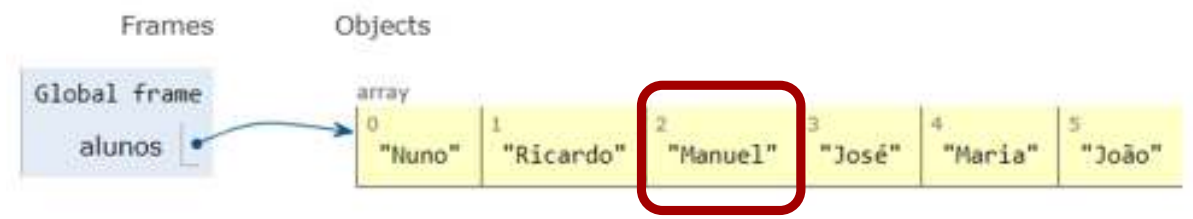
- Todos os tipos primitivos definem valores **imutáveis**
  - **O que não é o caso dos arrays!**

```
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 e 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);
```

propriedade

```
Nuno
Ricardo
6
```

```
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
constructor	Returns the function that created the Array object's prototype
length	Sets or returns the number of elements in an array
prototype	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*

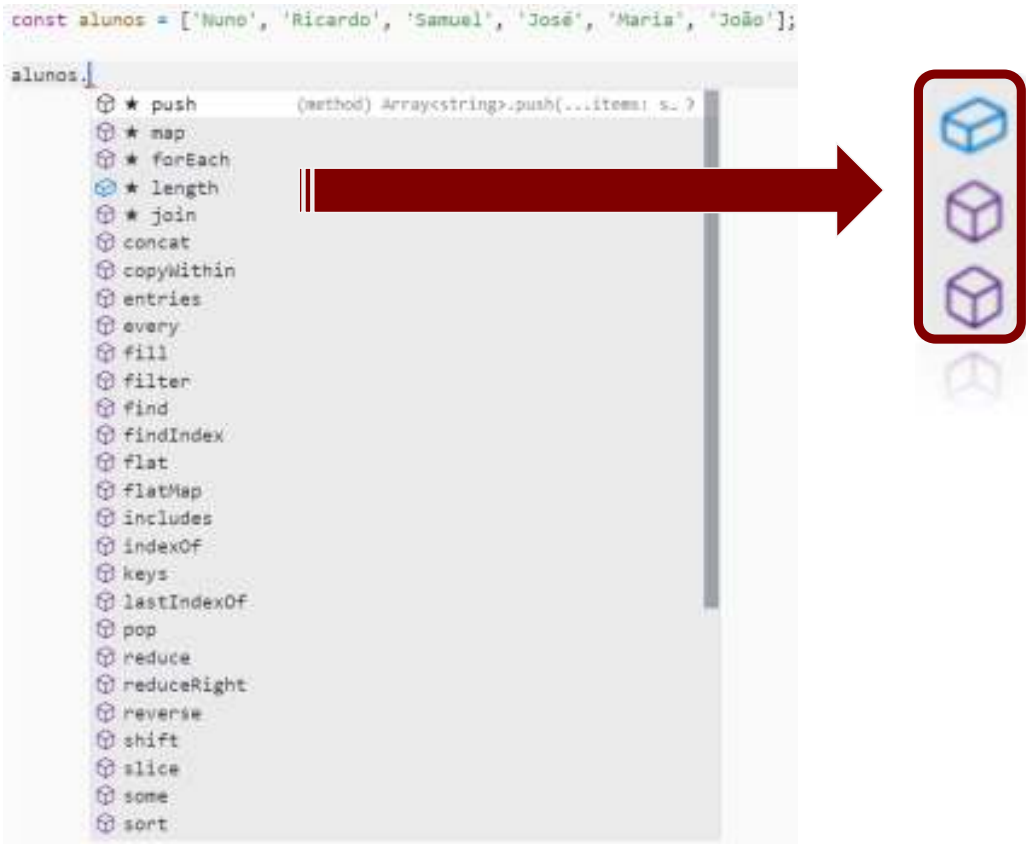
Arrays > Métodos

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

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

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

Arrays > Métodos



Arrays > Métodos > *push*

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

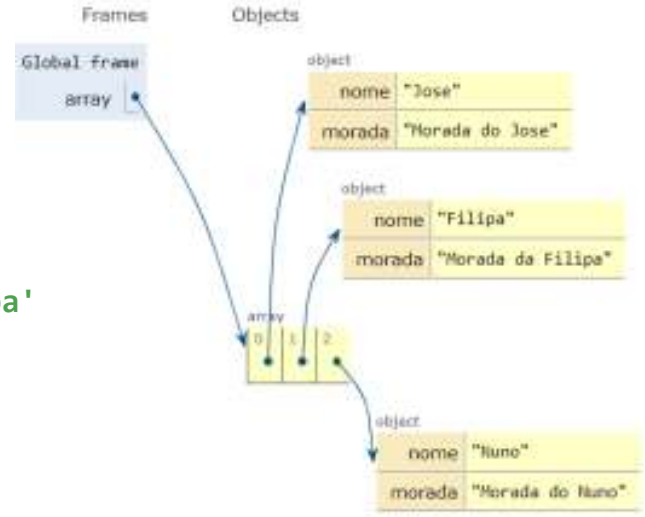
```
▼ (5) ['Nuno', 'Ricardo', 'Samuel', 'José', 'Maria']
    0: "Nuno"
    1: "Ricardo"
    2: "Samuel"
    3: "José"
    4: "Maria"
    length: 5
  ▶ [[Prototype]]: Array(0)
```

```
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: {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
                                                                           scrip
▼(8) ['Carlos', 'Nuno', 'Ricardo', 'Samuel', 'José', 'Maria', 'João', 'Afonso']
    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'];
```

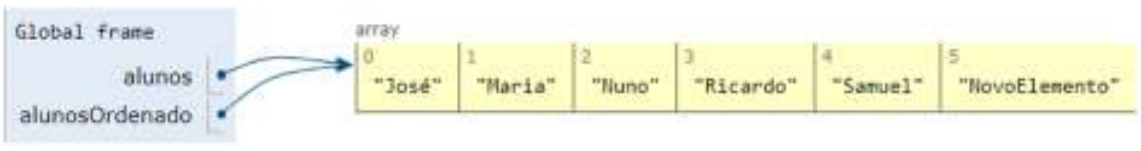
```
let alunosOrdenado = alunos.sort();
alunosOrdenado.push("NovoElemento");
console.table(alunos);
console.table(alunosOrdenado);
```

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.

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

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

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

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

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 > *map*

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

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

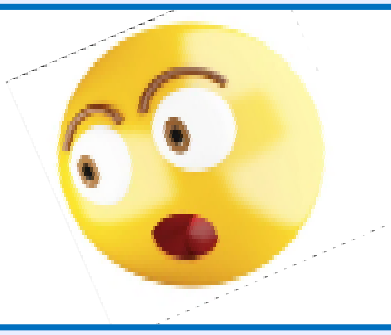
(index)	Value
0	15
1	18
2	21
3	24

```
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*


```
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
```

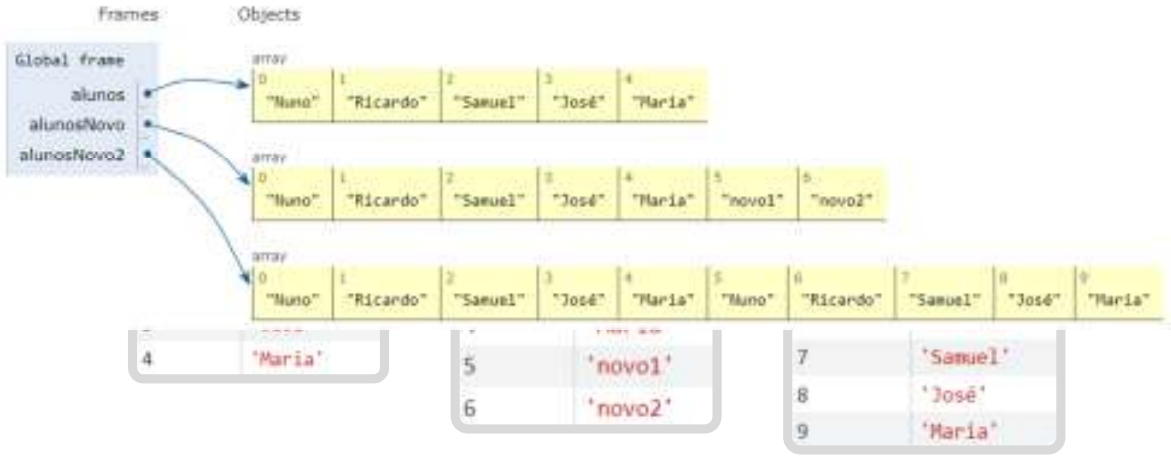
```
const nome='José'
if (alunos.includes(nome))
    console.log(`O ${nome} é aluno de Linguagens Script!`);
else
    console.log(`O ${nome} é não aluno de Linguagens Script!`);
```

```
▸ (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]
```



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 variaveis?

```
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) {
  // Codigo
}
```

```
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*

JavaScript

```
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*

JavaScript

```
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
```


JavaScript

</Arrays>