

JavaScript

ARRAY





Array

Array Accessor API

Array Iteration API

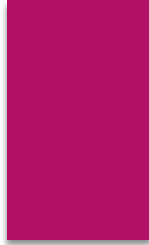
Array Mutator API

Array

Um Array é apenas um **objeto** que oferece operações para **acessar** e **manipular** suas propriedades

```
// A primeira forma é através dos colchetes  
const languages = []; console.log(languages);
```

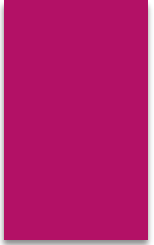
```
// É considerado um objeto  
console.log(typeof languages);
```



```
// Vamos criar e inicializar o array
const languages = ["Python", "Javascript", "R"];
console.log(languages);
```

```
// Podemos criar pela função construtora
const languages = new Array ();
console.log(languages);
```

```
// Vamos criar e inicializar pela função construtora
const languages = new Array ("Python", "Javascript", "R");
console.log(languages);
// atente a mudança de colchetes para parenteses
```



É possível **inicializar** passando **apenas**
um Number para a **função construtora**

```
// Cuidado ao passa Numbers para o Array
const numeros = new Array (1, 2, 3);
console.log(numeros);
// Com vários numeros tudo certo

// Veja o detalhe
const numeros = new Array (10);
console.log(numeros);
// é criado um array com 10 posições vazias.
```

A propriedade `length` indica a quantidade de elementos que existem dentro do Array

```
// Length - Mostra a quantidade de elementos
```

```
const timeUnits = [];  
  timeUnits[0] = "minute";  
  timeUnits[1] = "hour";  
  timeUnits[2] = "day";  
console.log(timeUnits.length);
```

```
// Cuidado! Os elementos vazios dentro  
// do Array não são considerado no length  
// o delete não é indicado
```

```
const timeUnits = [];  
  timeUnits[0] = "minute";  
  timeUnits[1] = "hour";  
  timeUnits[2] = "day";  
console.log(timeUnits);  
console.log(timeUnits.length);  
delete timeUnits[1];  
console.log(timeUnits);  
console.log(timeUnits.length);
```

Questão para Certificação JS

```
const languages = [];  
  languages[0] = "Python";  
  languages[10] = "Javascript";  
  languages[20] = "R";  
console.log(languages);  
console.log(languages.length);  
  
// Existem 3 elementos dentro mas em um For  
// não funcionaria, precisaria testar undefined 😞
```

Quantas posições tem esse Array?

Veremos no terminal do **node**

Array **Accessor** API

Os **accessor methods** quando invocados retornam informações específicas sobre o array

indexOf: Retorna a posição do primeiro elemento encontrado

lastIndexOf: Retorna a posição do último elemento encontrado

includes: Retorna true se o elemento existir

concat: Retorna um novo array resultante da concatenação de um ou mais arrays

slice: Retorna partes de um determinado array de acordo com a posição de início e fim

join: Converte o array para uma String, juntando os elementos com base em um separador

indexOf e lastIndexOf

```
// indexOf - posição do primeiro elemento encontrado
// lastIndexOf - posição do ultimo elemento encontrado
const languages = ["Python", "C", "Java"];
console.log(languages.indexOf("Python"));
console.log(languages.lastIndexOf("C"));
console.log(languages.indexOf("JavaScript"));

// Mesmo elemento encontrado 2 vezes
// em posições diferentes
const languages = ["Python", "C", "Java", "Python"];
console.log(languages.indexOf("Python"));
console.log(languages.lastIndexOf("Python"));

// Buscando elemento INExistente
console.log(languages.indexOf("Javascript"));
// retorna -1
```

includes

```
// includes - Retorna true se o elemento existir  
// dentro de um array (true or false)
```

```
const languages = ["Python", "C", "Java"];  
console.log(languages.includes("Python"));  
console.log(languages.includes("C"));  
console.log(languages.includes("JavaScript"));
```

concat

```
// Retorna um novo array resultante da  
// concatenação de um ou mais arrays
```

```
const ooLanguages = ["Smalltalk", "C++", "Simula"];  
const functionalLanguages = ["Haskell", "Scheme"];  
console.log(ooLanguages.concat(functionalLanguages));  
console.log(ooLanguages); console.log(functionalLanguages);  
// outra maneira, criando um novo array com o resultado  
const banana = [].concat(ooLanguages,functionalLanguages);  
console.log(banana)
```

slice

```
// slice - Retorna partes de um determinado array
// de acordo com a posição de início e fim
// SEMPRE N-1
const languages = ["Smalltalk", "C++", "Simula", "Haskell", "Scheme"];
console.log(languages.slice(0, 2));
console.log(languages.slice(2, 4));
console.log(languages.slice(1)); // DESTE ELEMENTO ATÉ O FINAL
```

join

```
// Join Converte o array para uma String
// juntando os elementos com base em um separador
const languages = ["Smalltalk", "C++", "Simula", "Haskell", "Scheme"];
console.log(languages.join(","));
console.log(languages.join(";"));
console.log(languages.join(" "));
// UTIL PARA EXPORTAR AQUIVOS
```

Array Iteration API

Os **iteration methods** quando invocados iteram sobre os elementos do array

Array Iteration API

forEach: Executa a função passada por parâmetro para cada elemento

filter: Retorna um novo array contendo somente os elementos que retornaram true na função passada por parâmetro

find: Retorna o primeiro elemento que retornou true na função passada por parâmetro

some: Retorna true se um ou mais elementos retornaram true na função passada por parâmetro

every: Retorna true se todos os elementos retornaram true na função passada por parâmetro

map: Retorna um novo array com base no retorno da função passada por parâmetro

reduce: Retorna um valor com base no retorno da função passada por parâmetro

forEach

```
// forEach - percorre cada elemento
// Executa a função passada por parâmetro para cada elemento

const frameworks = ["Angular.js", "Ember.js", "Vue.js"];
frameworks.forEach(framework => console.log(framework));
// Podemos substituir a função por uma
// ARROW FUNCTION - próxima aula vamos ver isso
```

filter

```
//Retorna um novo array contendo somente os elementos
// que retornaram true na função passada por parâmetro
const frameworks = [
  {
    name: "Angular.js",
    contributors: 1598 },
  {
    name: "Ember.js",
    contributors: 746},
  {
    name: "Vue.js",
    contributors: 240}
];
const result = frameworks.filter(function (framework) {
  return framework.contributors < 1000; });
console.log(result);
```

find

```
// find - Retorna o primeiro elemento que retornou
// true na função passada por parâmetro
const frameworks = [
  {
    name: "Angular.js",
    contributors: 1598 },
  {
    name: "Ember.js",
    contributors: 746 },
  {
    name: "Vue.js",
    contributors: 240 }
];
const result = frameworks.find(function (framework) {
  return framework.name === "Angular.js";
});
console.log(result);
```


some

```
//some -Retorna true se um ou mais elementos retornaram
// true na função passada por parâmetro
const frameworks = [
  {
    name: "Angular.js",
    contributors: 1598 },
  {
    name: "Ember.js",
    contributors: 746 },
  {
    name: "Vue.js",
    contributors: 240 }
];
const result = frameworks.some(function (framework) {
  return framework.name.includes("js");
});
console.log(result);
// Conjunto checkbox, você quer mostrar um botão
// deletar se 1 ou mais estiverem marcados
```

every

```
//every -Retorna true se todos os elementos retornarem
// true na função passada por parâmetro
const frameworks = [
  {
    name: "Angular.js",
    contributors: 1598 },
  {
    name: "Ember.js",
    contributors: 746 },
  {
    name: "Vue.js",
    contributors: 240 }
];
const result = frameworks.every(function (framework) {
  return framework.contributors > 1000;
});
console.log(result);
// Conjunto checkbox, você quer mostrar um botão
// deletar se 1 ou mais estiverem marcados
```

map

```
// map - Retorna um novo array com base no retorno
// da função passada por parâmetro
const frameworks = [
  {
    name: "Angular.js",
    contributors: 1598 },
  {
    name: "Ember.js",
    contributors: 746 },
  {
    name: "Vue.js",
    contributors: 240 }
];
const result = frameworks.map(function (framework) {
  return framework.name;
});
//const result = frameworks.map((framework) => framework.name);
console.log(result);
// é como retornar um novo array com o que você quer
// no caso acima, os nomes dos frameworks
```

reduce

```
// reduce - Retorna um valor com base no retorno
// da função passada por parâmetro
const frameworks = [
  {
    name: "Angular.js",
    contributors: 1598 },
  {
    name: "Ember.js",
    contributors: 746 },
  {
    name: "Vue.js",
    contributors: 240 }
];
const result = frameworks.reduce(function (total, framework) {
  return total + framework.contributors;
}, 0);
console.log(result);
// 2 parametros (total = 0) (soma contributors)
// comum em operações matemáticas... preços e numeros
// pode ser substituído por forEach ... SIM
```

Array **Mutator** API

Os **mutator methods** quando invocados
modificam o array

Array **Mutator** API

push: Adiciona um elemento no final

pop: Remove um elemento do final

unshift: Adiciona um elemento no início

shift: Remove um elemento do início

splice: Remove, substitui ou adiciona um ou mais elementos em uma determinada posição

sort: Ordena os elementos de acordo com a função de ordenação

reverse: Inverte a ordem dos elementos

fill: Preenche os elementos de acordo com a posição de início e fim

push e pop

```
// push: Adiciona um elemento no final  
// pop: Remove um elemento do final
```

```
const languages = ["Python", "C", "Java"];  
console.log(languages);
```

```
console.log(languages.push("Ruby"));  
console.log(languages.push("Go"))  
console.log(languages);
```

```
console.log(languages.pop());  
console.log(languages.pop());  
console.log(languages);
```

unshift e shift

```
// unshift: Adiciona um elemento no início  
// shift: Remove um elemento do início
```

```
const languages = ["Python", "C", "Java"];  
console.log(languages);
```

```
console.log(languages.unshift("Ruby"));  
console.log(languages.unshift("Go"));  
console.log(languages);
```

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


splice

```
// splice: Remove, substitui ou adiciona  
// um ou mais elementos em uma determinada posição  
// não use delete... use splice
```

```
const languages = ["Python", "C", "Java"];  
console.log(languages);
```

```
console.log(languages.splice(1, 1)); //(posição e qtd elementos) - removeu 1 elemento C  
console.log(languages);
```

```
console.log(languages.splice(1, 0, "C++", "C#")); // inseriu 2 elementos - empurrando  
console.log(languages);
```

```
console.log(languages.splice(1, 2, "C")); // removeu 2 e inclui C  
console.log(languages);
```

sort

```
// sort: Ordena os elementos de acordo com a função de ordenação  
// ordenando por a - b (crescente)  
// retorna 1 ou -1
```

```
const frameworks = [  
  {  
    name: "Angular.js",  
    year: 1991},  
  {  
    name: "Ember.js",  
    year: 1972},  
  {  
    name: "Vue.js",  
    year: 1995}  
];  
languages.sort(function (a, b) {  
  return a.year - b.year;  
});  
console.log(languages);
```

reverse

```
///// reverse: Inverte a ordem dos elementos
```

```
const languages = ["Python", "C", "Java"];
```

```
languages.reverse();  
console.log(languages);
```

```
languages.reverse();  
console.log(languages);
```

fill

```
// fill: Preenche os elementos de acordo com a
// posição de início e fim
const languages = ["Python", "C", "Java"];
languages.fill("JavaScript"); // se não especificar preenche todas as posições
console.log(languages);

const languages = ["Python", "C", "Java"];
languages.fill("JavaScript", 1); // apenas 2 parametros, preenche a partir dessa posição
console.log(languages);

const languages = ["Python", "C", "Java"];
languages.fill("JavaScript", 0, 2); // 3 parametros, a partir da posição, qtd (N -1)
console.log(languages);
```