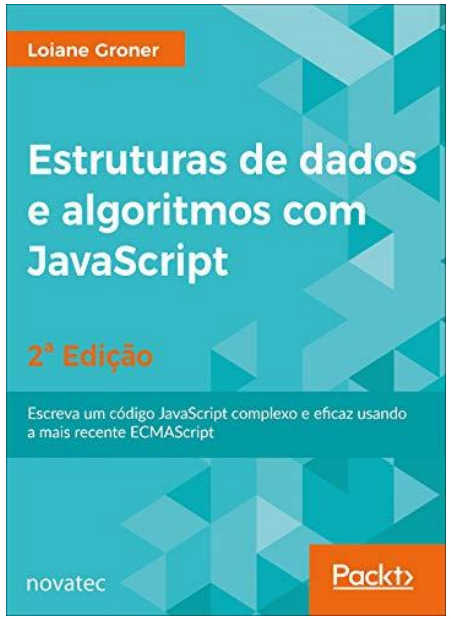
Arrays

(Vetores unidimensionais e Matrizes bidimensionais e multidimensionais)

Referências Bibliográficas:

GRONER, Loiane. **Estruturas de dados e algoritmos com JavaScript**: Escreva um código JavaScript complexo e eficaz usando a mais recente ECMAScript. 2ª ed. São Paulo: Novatec, 2019.



SIF005 - Estrutura de Dados -

Prof. Dr. Anderson Sena – anderson.sena@iesb.edu.br

Definição básica

- Um **array** é a estrutura de dados mais simples possível em memória.

- Um **array** armazena valores que são todos do mesmo tipo (homogêneo), sequencialmente (como uma lista de valores).

- Embora o **JavaScript** nos permita criar arrays com valores de tipos distintos.

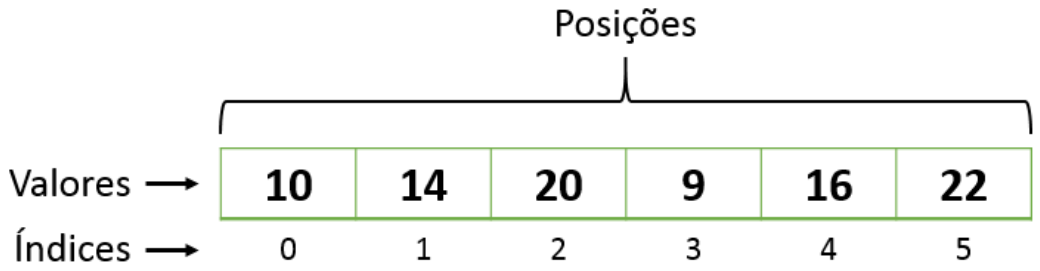
```
const numbers = [1,2,3,[4,5,6]];

console.log(numbers.length);
// Saída: 4
```


Fonte: Loiane Groner, 2019. Pág. 79

Definição básica

- Um **arranjo** (em inglês **array**) é uma estrutura de dados que armazena uma coleção de elementos de tal forma que cada um dos elementos possa ser identificado por, pelo menos, um índice (chave), geralmente do mesmo tamanho e tipo de dados (homogêneos).
- A posição é dada por um **índice**, também chamado de **subscrição**.
- Essa estrutura de dados também é conhecida como **variável indexada**, **vetor** (para arranjos unidimensionais) e **matriz** (para arranjos bidimensionais).
- Geralmente, arranjos unidimensionais e bidimensionais são os mais comuns



Fonte: Wikipedia https://pt.wikipedia.org/wiki/Arranjo_(computa%C3%A7%C3%A3o), acesso em 22fev2021

Por que devemos usar arrays?

- Vamos supor que precisamos armazenar a **temperatura média** de cada mês do ano para a cidade em que vivemos.
- Poderíamos usar algo semelhante a este código em JavaScript, para armazenar essas informações:

```
1    const averageTempJan = 31;
2    const averageTempFeb = 30.5;
3    const averageTempMar = 28.5;
4    const averageTempAbr = 27.8;
5    const averageTempMay = 25.6;
6
```

Por que devemos usar arrays?

- No entanto, essa não é a melhor abordagem. O que ocorreria, se precisássemos armazenar a temperatura média para mais de um ano?

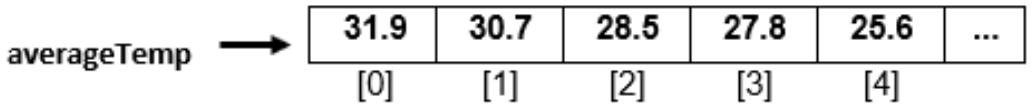
- Esse seria um motivo para o qual os arrays foram criados. Poderíamos representar melhor da seguinte forma:

```
const averageTemp = [];
averageTemp[0] = 31.9;
averageTemp[1] = 30.7;
averageTemp[2] = 28.5;
averageTemp[3] = 27.8;
averageTemp[4] = 25.6;

console.log('Conteúdo do array:', averageTemp);
console.log('Média de temperatura de março: ', averageTemp[2]);
console.log('Média de temperatura de janeiro: ', averageTemp[0]);
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.js"
Conteúdo do array: [ 31.9, 30.7, 28.5, 27.8, 25.6 ]
Média de temperatura de março:  28.5
Média de temperatura de janeiro:  31.9
```

Também podemos representar o array **averageTemp** graficamente:



Criando e inicializando arrays

- Declarar, criar e inicializar um array em JavaScript é realmente simples:

```
1   // Podemos apenas declarar e instanciar um novo array usando a palavra reservada new
2   let daysOfWeek = new Array();
3
4   //Alem disso, usando a palavra reservada new, podemos criar um array especificando seu tamanho
5   daysOfWeek = new Array(7);
6
7   // Outra opção, é passar os elementos do array diretamente para o seu construtor
8   daysOfWeek = new Array('Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday');
9
10  // Contudo, usar a palavra reservada new não é considerada a melhor prática.
11  // Se quisermos criar um array em JavaScript, podemos atribuir colchetes vazios []
12  let daysOfWeek = [];
13
14  // também podemos inicializar o array com alguns elementos assim
15  let daysOfWeek = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday'];
16
17  // Se quisermos saber quantos elementos há no array (seu tamanho) com a propriedade lenght
18  console.log(daysOfWeek.length);
19
```


Fonte: Loiane Groner, 2019. Pág. 81

Acessando elementos e fazendo iteração

- Para acessar uma posição, podemos usar **colchetes**, passando o **índice** da posição.
- Para percorrer o **array** com um **laço** e exibir os elementos, **começando pelo índice zero**:

```
1  // também podemos inicializar o array com alguns elementos assim
2  let daysOfWeek = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday'];
3
4  // percorrendo cada um dos elementos do array
5  for (let i = 0; i < daysOfWeek.length; i++){
6      console.log(daysOfWeek[i]);
7  }
8
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE Code

```
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.js"
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
```


Fonte: Loiane Groner, 2019. Pág. 81

Acessando elementos e fazendo iteração

Outro exemplo com os 20 primeiros números da sequência Fibonacci (*soma os dois anteriores*):

```
1   // armazenando a série fibonacci no array com 20 elementos
2   const fibonacci = [];
3   fibonacci[0] = 1;
4   fibonacci[1] = 1;
5   for (let i = 2; i < 20; i++) {
6       fibonacci[i] = fibonacci[i - 1] + fibonacci[i - 2];
7   }
8   // se quiser imprimir no console, um elemento ao lado do outro,
9   // faça a conversão em string, concatenando com o método join.
10  console.log(fibonacci.join('  '));
11  // percorrendo o array e mostrando no console cada elemento ficonacci
12  for (let i = 0; i < fibonacci.length; i++) {
13      console.log(fibonacci[i]);
14  }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
1  1  2  3  5  8  13  21  34  55  89  144  233  377  610  987  1597  2584  4181  6765
1
1
2
3
5
8
13
21
```

Acrescentando elementos

- Em JavaScript, um **array** é um **objeto mutável**.
- Ele crescerá **dinamicamente** à medida que novos elementos forem **adicionados**.
- Em várias outras linguagens, por exemplo, em C e em Java, **é preciso** determinar o **tamanho** do array e, caso haja necessidade de adicionar mais elementos, um array totalmente novo deverá ser criado.

```
1   // acrescentar e remover elementos de um array não é difícil
2   // vamos considerar o seguinte array
3   let numbers = [0, 1, 2, 3, 4, 5 , 6, 7, 8, 9];
4
5   // se quisermos acrescentar mais um elemento, é só referenciar a
6   // última posição livre e atribuir um valor
7   numbers[numbers.length] = 10;
8
9   console.log(numbers);
10
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.js"
[
  0, 1, 2, 3, 4,
  5, 6, 7, 8, 9,
  10
]
```

Acrescentando elementos com o método **push**

A API de JavaScript também tem um método chamado **push()**, que permite acrescentar novos elementos no final de um **array:**

O array **numbers** exibirá os números de 0 a 13 como saída:

```
1   // acrescentar e remover elementos de um array não é difícil
2   // vamos considerar o seguinte array
3   let numbers = [0, 1, 2, 3, 4, 5 , 6, 7, 8, 9];
4
5   // se quisermos acrescentar mais um elemento, é só referenciar a
6   // última posição livre e atribuir um valor
7   numbers[numbers.length] = 10;
8   // acrescentando elementos no final do array com o método push
9   numbers.push(11);
10  numbers.push(12, 13);
11
12  console.log(numbers);
13  // se quiser converter em uma string é só concatenar com uma vírgula
14  // para apresentar na console
15  console.log(numbers.join(', '));
16
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.js"
[
  0,  1, 2, 3,  4,  5,
  6,  7, 8, 9, 10, 11,
 12, 13
]
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
```

Inserindo um elemento na primeira posição

- ✓ Vamos agora inserir na primeira posição o número **-1**
- ✓ Teremos que deslocar todos os elementos pra direita.

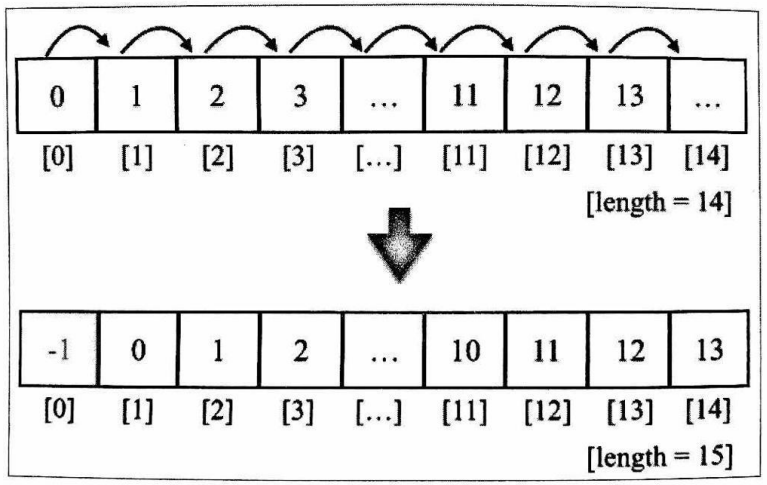


Figura 3.2

Fonte: Loiane Groner, 2019. Pág. 83

```
2   // vamos considerar o seguinte array
3   let numbers = [0, 1, 2, 3, 4, 5 , 6, 7, 8, 9];
4   // se quisermos acrescentar mais um elemento, é só referenciar a
5   // última posição livre e atribuir um valor
6   numbers[numbers.length] = 10;
7   // acrescentando elementos no final do array com o método push
8   numbers.push(11);
9   numbers.push(12, 13);
10  // devemos deixar a primeira posição livre, deslocando
11  // todos os elementos para a direita.
12  function insertFirstPosition(value) {
13      for (let i = numbers.length; i>=0; i--)  {
14          numbers[i] = numbers[i - 1];
15      }
16      numbers[0] = value
17  };
18  // chamando a função
19  insertFirstPosition(-1);
20  // se quiser converter em uma string é só concatenar com uma vírgula
21  // para apresentar na console
22  console.log(numbers.join(', '));
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.js"
-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
```

unshift( ) insere no início do Array

- ✓ A classe **array** de JavaScript também tem um método chamado **unshift**, que insere no início do array os valores passados como argumentos para o método.
- ✓ A lógica interna tem o mesmo comportamento da função **insertFirstPosition**.

```
21   // vamos considerar o seguinte array
22   let numbers = [0, 1, 2, 3, 4, 5 , 6, 7, 8, 9];
23   // inserindo no início com o método unshift
24   numbers.unshift(-1);
25   numbers.unshift(-2);
26   numbers.unshift(-4, -3);
27   // para apresentar na console
28   console.log(numbers.join(', '));
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempC
-4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

pop( ) remove elementos do final do Array

✓ Para remover um valor do final do array, podemos utilizar o método **pop**

Os métodos **push**( ) e **pop**( ) permitem que um array emule uma estrutura de dados **stack** (pilha) básica, que será nosso assunto mais adiante.

```
21  // vamos considerar o seguinte array
22  let numbers = [0, 1, 2, 3, 4, 5 , 6, 7, 8, 9];
23  // inserindo no início com o método unshift
24  numbers.unshift(-1);
25  numbers.unshift(-2);
26  numbers.unshift(-4, -3);
27  // Removendo um elemento do final do array
28  numbers.pop();
29  // para apresentar na console
30  console.log(numbers.join(', '));
31  // Removendo mais um elemento do final do array
32  numbers.pop();
33  console.log(numbers.join(', '));
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCod
-4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8
```

Removendo um elemento da primeira posição

✓ Para remover um valor do início do array, podemos usar o seguinte código:

```
1   // vamos considerar o seguinte array
2   let numbers = [0, 1, 2, 3, 4, 5 , 6, 7, 8, 9];
3   //Para remover um valor do início do array, podemos:
4   for (var i=0; i < numbers.length; i++) {
5       numbers[i] = numbers[i+1];
6   }
7   // Deslocamos sobrescrevendo todos os elementos pra esquerda
8   // Entretanto o array numbers continua com tamanho (lenght) 10
9   // significa que a última posição está com valor undefined
10  console.log(numbers);
11
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.js"
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, undefined ]
```

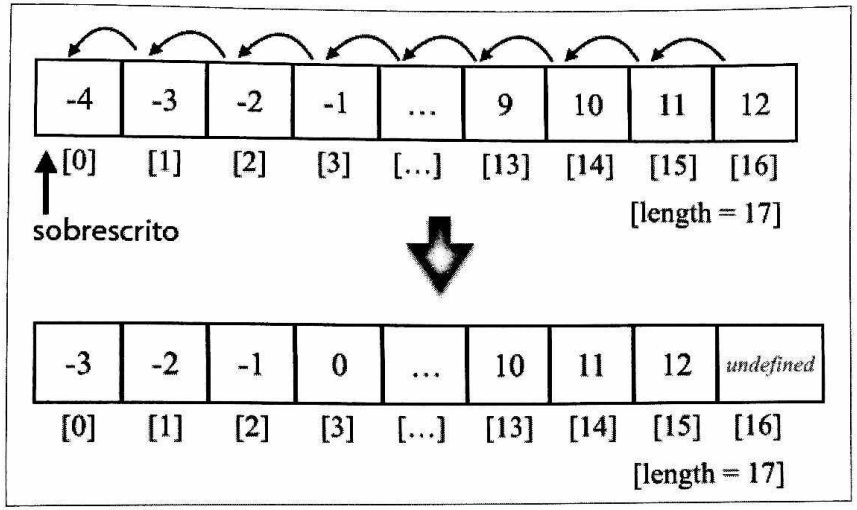


Figura 3.3

✓ Simplesmente sobrescrevemos os valores originais do array, sem de fato tê-los removido.

Fonte: Loiane Groner, 2019. Pág. 85

Removendo um elemento da primeira posição

✓ Para remover o valor do array, podemos também criar um método removeFirstPosition

```
let numbers = [-4, -3, -2, -1, 0, 1, 2, 3, 4, 5 , 6, 7, 8, 9, 10, 11, 12];
// copiando todos os valores para outro array
// diferentes de undefined do array original
Array.prototype.reIndex = function(myArray) {
 const newArray = [];
 for (var i = 0; i < myArray.length; i++) {
     if (myArray[i] !== undefined) {
         // console.log(numbers[i]);
         newArray.push(myArray[i]);
     }
 }
 return newArray;
}
//remove a primeira posição manualmente e executa reIndex
Array.prototype.removeFirstPosition = function() {
    for (var i=0; i < this.length; i++) {
        this[i] = this[i + 1];
    }
    return this.reIndex(this);
}
numbers = numbers.removeFirstPosition();
// apresentando o array numbers após a execução do método
console.log(numbers);
```

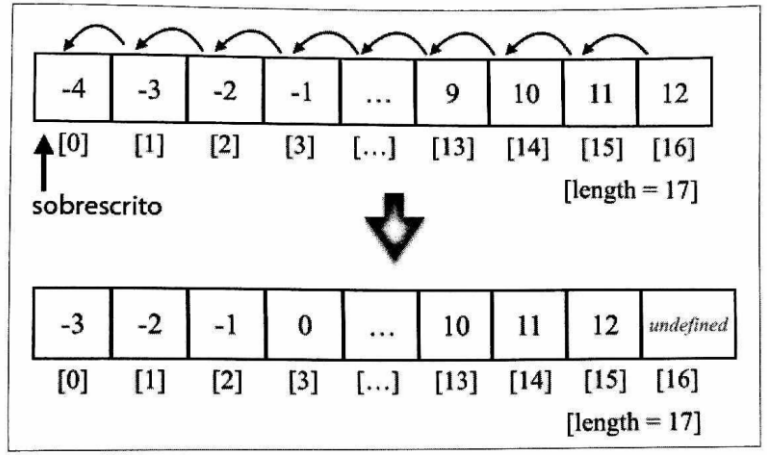


Figura 3.3

```
23  console.log(numbers);
24
PROBLEMS  OUTPUT  TERMINAL  DEB
[Running] node "c:\Users\admi
[
  -3, -2, -1,  0, 1, 2,
   3,  4,  5,  6, 7, 8,
   9, 10, 11, 12
]
```

Fonte: Loiane Groner, 2019. Pág. 85

Removendo um elemento da primeira posição

- ✓ Para remover um valor do início do array, podemos usar o método **shift ( )**

Os métodos **shift**( ) e **unshift**( ) permitem que um array emule uma estrutura de dados **queue** (fila) básica, que será nosso assunto mais adiante.

```
32  // vamos considerar o seguinte array
33  let numbers = [0, 1, 2, 3, 4, 5 , 6, 7, 8, 9];
34  // removendo um elemento da primeira posição
35  numbers.shift();
36  // mostrando o conteúdo do array
37  console.log('O tamanho do array agora e: ' + numbers.length);
38  console.log(numbers.join(', '));
39  // removendo um elemento da primeira posição
40  numbers.shift();
41  console.log('O tamanho do array agora e: ' + numbers.length);
42  console.log(numbers.join(', '));
43
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.js
O tamanho do array agora e: 9
1, 2, 3, 4, 5, 6, 7, 8, 9
O tamanho do array agora e: 8
2, 3, 4, 5, 6, 7, 8, 9
```

Fonte: Loiane Groner, 2019. Pág. 85

Adicionando ou removendo elementos de uma posição específica

- O método **splice( )** pode ser usado para remover um elemento de um array, simplesmente especificando a posição (índice) a partir do qual queremos fazer a remoção, e a quantidade de elementos que queremos remover.

- Usamos sempre os métodos **splice( )**, **pop( )** ou **shift( )** para remover elementos, pois eles fazem a reindexação dos elementos.

```
1   // vamos considerar o seguinte array
2   let numbers = [0, 1, 2, 3, 4, 5 , 6, 7, 8, 9];
3   // removendo três elementos a partir da posição 5
4   numbers.splice(5, 3);
5   // mostrando o conteúdo do array
6   console.log(numbers.join(', '));
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeR
0, 1, 2, 3, 4, 8, 9
```

Fonte: Loiane Groner, 2019. Pág. 86

Adicionando ou removendo elementos de uma posição específica

Se quisermos **inserir** os números 5, 6 e 7 de volta no array, a partir da posição 5, podemos usar novamente o método **splice**.

- ✓ O primeiro argumento é o índice a partir do qual queremos remover ou inserir elementos.
- ✓ O segundo argumento é a quantidade de elementos que queremos remover.
- ✓ Do terceiro em diante, temos os valores que gostaríamos de inserir no array.

```
1   // vamos considerar o seguinte array
2   let numbers = [0, 1, 2, 3, 4, 5 , 6, 7, 8, 9];
3   // removendo três elementos a partir da posição 5
4   numbers.splice(5, 3);
5   // mostrando o conteúdo do array
6   console.log(numbers.join(', '));
7   // reinserindo os três elementos (5, 6, 7) a partir da posição 5
8   // o parâmetro 0 será a quantidade de números excluídos
9   numbers.splice(5, 0, 5, 6, 7);
10  // mostrando o conteúdo do array
11  console.log(numbers.join(', '));
12
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.js"
0, 1, 2, 3, 4, 8, 9
0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

Fonte: Loiane Groner, 2019. Pág. 87

Arrays bidimensionais e multidimensionais

✓ Suponha que precisamos medir a temperatura de hora em hora durante alguns dias.

```
1   // é possível usar arrays para armazenar temperaturas de dois dias.
2   var averageTempDay1 = [36.5, 28, 30, 25, 18, 40];
3   var averageTempDay2 = [19, 22.5, 29, 31, 23.5, 32.5];
4
```

✓ No entanto, essa não é a melhor abordagem! Uma **matriz** (array bidimensional ou um *array de arrays*) pode ser usada:

A linguagem JavaScript aceita apenas arrays unidimensionais; ela não tem suporte para **matrizes**. Contudo, podemos implementar qualquer array multidimensional, usando **array de arrays**, como nesse código:

```
6   var averageTemp = [];
7   averageTemp[0] = [36.5, 28, 30, 25, 18, 40];
8   averageTemp[1] = [19, 22.5, 29, 31, 23.5, 32.5];
9   // para mostrar a última temperatura
10  console.log(averageTemp[1][5]);
11
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCod
32.5
```


Fonte: Loiane Groner, 2019. Pág. 87

Iterando pelos elementos de arrays bidimensionais

✓ Se quisermos ver a saída da matriz, podemos criar uma função genérica para fazer o log:

ed-js > js > JS array.js > ...

```
1   var averageTempDay1 = [36.5, 28, 30, 25, 18, 40];
2   var averageTempDay2 = [19, 22.5, 29, 31, 23.5, 32.5];
3
4   var averageTemp = [];
5   averageTemp[0] = averageTempDay1;
6   averageTemp[1] = averageTempDay2;
7
8   function printMatrix(myMatrix) {
9       for (let i = 0; i < myMatrix.length; i++)  {
10          for (let j = 0; j < myMatrix[i].length; j++) {
11              console.log(myMatrix[i][j]);
12          }
13      }
14  }
15
16  printMatrix(averageTemp);
```

```
16  printMatrix(averageTemp);
```

PROBLEMS OUTPUT TERMINAL DEBUG C

```
[Running] node "c:\Users\admin\De
36.5
28
30
25
18
40
19
22.5
29
31
23.5
32.5
```


Fonte: Loiane Groner, 2019. Pág. 88

Iterando pelos elementos de arrays bidimensionais

- ✓ Para exibir um array bidimensional no console do navegador, podemos usar também a instrução console.table(averageTemp).
- ✓ Com ela, teremos uma saída mais elegante para o usuário.

ed-js > js > JS array.js > ...

```
1   var averageTempDay1 = [36.5, 28, 30, 25, 18, 40];
2   var averageTempDay2 = [19, 22.5, 29, 31, 23.5, 32.5];
3
4   var averageTemp = [];
5   averageTemp[0] = averageTempDay1;
6   averageTemp[1] = averageTempDay2;
7
8   function printMatrix(myMatrix) {
9       for (let i = 0; i < myMatrix.length; i++)  {
10          for (let j = 0; j < myMatrix[i].length; j++) {
11              console.log(myMatrix[i][j]);
12          }
13      }
14  }
15  //printMatrix(averageTemp);
16  console.table(averageTemp);
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunner
```

(index)	0	1	2	3	4	5
0	36.5	28	30	25	18	40
1	19	22.5	29	31	23.5	32.5

Fonte: Loiane Groner, 2019. Pág. 89

Arrays multidimensionais

- ✓ É possível também trabalhar com arrays multidimensionais em JavaScript.
- ✓ Por exemplo, vamos criar uma matriz 3x3.
- ✓ Cada célula contém a soma i (linha) + j (coluna) + z (profundidade) da matriz, deste modo:

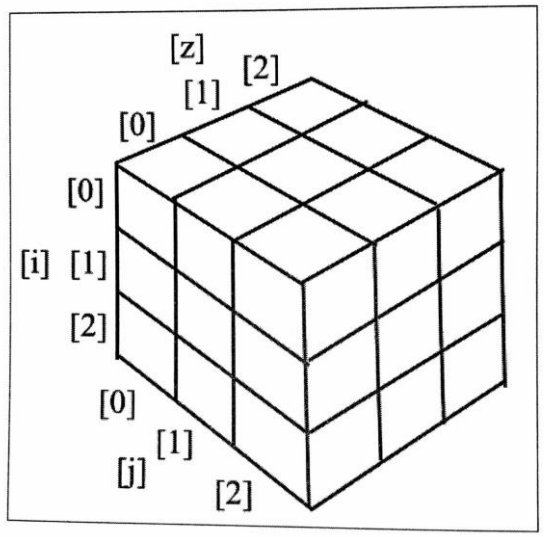


Figura 3.5

ed-js > js > JS array.js > ...

```
1   const matrix3x3x3 = [];
2
3   for (var i = 0; i < 3; i++){
4       matrix3x3x3[i] = []; //precisamos inicializar cada array
5       for (var j = 0; j < 3; j++ ){
6           matrix3x3x3[i][j] = [];
7           for (var z = 0; z < 3; z++) {
8               matrix3x3x3[i][j][z] = i + j + z;
9           }
10      }
11  }
12  console.table(matrix3x3x3);
13
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.
```

(index)	0	1	2
0	[0, 1, 2]	[1, 2, 3]	[2, 3, 4]
1	[1, 2, 3]	[2, 3, 4]	[3, 4, 5]
2	[2, 3, 4]	[3, 4, 5]	[4, 5, 6]

Fonte: Loiane Groner, 2019. Pág. 89

Referências para métodos de array em JavaScript

Os arrays em JavaScript são objetos modificados e possuem alguns métodos disponíveis pra uso:

Já discutimos os métodos **push**, **pop**, **shift**, **unshift** e **splice**.

Alguns desses métodos são muito úteis quando trabalhamos com **programação funcional**.

Método	Descrição
concat	Junta vários arrays e devolve uma cópia dos arrays concatenados.
every	Itera por todos os elementos do array, verificando uma condição desejada (função) até que false seja devolvido.
filter	Cria um array com todos os elementos avaliados com true pela função especificada.
forEach	Executa uma função específica em cada elemento do array.
join	Reúne todos os elementos do array em uma string.
indexOf	Pesquisa o array em busca de elementos específicos e devolve a sua posição.
lastIndexOf	Devolve a posição do último item do array que corresponda ao critério de pesquisa.
map	Cria outro array a partir de uma função que contém o critério/condição e devolve os elementos do array que correspondam ao critério.
reverse	Inverte o array, de modo que o último item se torne o primeiro, e vice-versa.
slice	Devolve um novo array a partir do índice especificado.
some	Itera por todos os elementos do array, verificando a condição desejada (função) até que true seja devolvido.
sort	Organiza o array em ordem alfabética ou de acordo com a função especificada.
toString	Devolve o array na forma de uma string.
valueOf	É semelhante ao método toString e devolve o array na forma de uma string.

Fonte: Loiane Groner, 2019. Pág. 91

Juntando vários arrays

- Considere um cenário em que você tenha arrays diferentes e precise juntar todos eles.
- Poderíamos fazer uma iteração em cada array e acrescentar cada um dos elementos no array final.
- Mas, felizmente, o JavaScript possui um método chamado **concat( )** capaz de fazer isso.
- Neste exemplo, **zero** será concatenado a **negativeNumbers**, e então **positiveNumbers** será concatenado no array resultante:

```
1   const zero = 0;
2   const positiveNumbers = [1, 2, 3];
3   const negativeNumbers = [-3, -2, -1];
4   let numbers = negativeNumbers.concat(zero, positiveNumbers);
5
6   console.log(numbers.join(', '));
7
```

```
PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.
-3, -2, -1, 0, 1, 2, 3
```


Fonte: Loiane Groner, 2019. Pág. 91

Funções de iteração

- Às vezes precisamos iterar pelos elementos de um array.
- Vimos que um laço pode ser usado para isso, por exemplo, a instrução for, conforme exemplos anteriores.
- A linguagem JavaScript tem alguns **métodos de iteração embutidos**, que podem ser usados com arrays.
- Para exemplificar, precisaremos de um array e de uma função. Usaremos um **array** com valores de 1 a 15, além de uma **função** que devolverá **true** se o número for múltiplo de 2 (par), e **false** caso contrário.

```
1   function isEven(x){
2       // devolve true e x for múltiplo de 2.
3       console.log(x);
4       return (x % 2 === 0) ? true : false;
5   }
6
7   let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
8
```

Fonte: Loiane Groner, 2019. Pág. 92

Iterando com o método every

- O método **every( )** itera pelos elementos do **array** até que a função devolva **false.**

numbers.every(isEven);

- Nesse caso, o nosso primeiro elemento do array **numbers** é **1**.

- O número 1 não é múltiplo de 2, pois é ímpar, portanto a função isEven devolverá **false** logo na leitura do primeiro elemento:

```
1   function isEven(x){
2       // devolve true e x for múltiplo de 2.
3       console.log(x);
4       return (x % 2 === 0) ? true : false;
5   }
6   let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
7   // Nesse caso, o nosso primeiro elemento do array numbers é 1.
8   // O número 1 não é múltiplo de 2, pois é ímpar, portanto a função
9   // isEven devolverá false logo na leitura do primeiro elemento
10  numbers.every(isEven);
11  // O método estático every() itera pelos elementos do array
12  // até que a função devolva false.
13
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ED-SEXTA\javaScript\tempCodeRunner
1
```

Iterando com o método some

- Temos também o método some( ), que apresenta o comportamento oposto ao método every( ).
- No entanto, o método some( ) itera pelos elementos do array **até** que a função devolva **true**.

numbers.some(isEven);

- Nesse caso, o primeiro número par em nosso array numbers é **2** (*o segundo elemento*).
- O primeiro elemento da iteração é o número 1; ele devolverá **false**.
- Então, o segundo elemento da iteração será o número 2, que devolverá **true**, e a interação **será interrompida**.

```
1   function isEven(x){
2       // devolve true e x for múltiplo de 2.
3       console.log(x);
4       return (x % 2 === 0) ? true : false;
5   }
6
7   let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
8
9   numbers.some(isEven);
10  // O primeiro elemento da iteração é o número 1; ele devolverá false.
11  // Então, o segundo elemento da iteração será 2, que devolverá true.
12  // Então a iteração será interrompida, pois o método some itera pelos
13  // elementos até que a função devolva true.
14
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ED-SEXTA\javaScript\tempCodeRunnerFil
1
2
```

Iterando com forEach

- Se precisarmos fazer a iteração em todo o array, independentemente de tudo mais, podemos usar a função forEach.
- O resultado será o mesmo que usar um laço for com o código da função dentro dele, assim:

numbers.forEach(x => console.log(x % 2 === 0));

```
1   let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
2
3   numbers.forEach( x => console.log( x % 2 === 0));
4
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ED-SEXTA\javaScript\tempCodeRunnerF
false
true
false
true
false
true
false
true
false
true
false
true
false
true
false
```

Usando map e filter

- A linguagem JavaScript também tem outros dois métodos iteradores que devolvem um novo array com um resultado.
- O primeiro é o método **map**:

```
const myMap = numbers.map(isEven);
```

- Ele armazena os resultados da função **isEven**, passada para o método map.
- Desse modo, podemos facilmente saber se um número é par ou não.

*Por exemplo, myMap[0] devolve **false** porque 1 não é par, e myMap[2] devolve **true** porque 2 é par.*

```
1   function isEven(x){
2       // devolve true e x for múltiplo de 2.
3       //console.log(x);
4       return (x % 2 === 0) ? true : false;
5   }
6
7   let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
8
9   const myMap = numbers.map(isEven);
10
11  console.log(myMap);
12
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ED-SEXTA\javaScript\tempCodeRunner
[
  false, true,  false,
  true,  false, true,
  false, true,  false,
  true,  false, true,
  false, true,  false
]
```

Usando map e filter

- Também temos o método **filter**, o qual devolve um **novo array** com os elementos para os quais a função devolveu **true**, assim:

```
const evenNumbers =
        numbers.filter(isEven);
```

- A array **evenNumbers** conterá os elementos que são múltiplos de 2: [2, 4, 6, 7, 10, 12, 14]

```
1   // Usaremos um array co os valores de 1 a 15,
2   // além de uma função que devolverá true para múltiplo de 2(par)
3   // e false caso contrário (ímpares)
4   let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
5
6   function isEven(x){
7       // devolve true e x for múltiplo de 2.
8       //console.log(x);
9       return (x % 2 === 0) ? true : false;
10  }
11  // o método filter devolve um novo arrau com os elementos para os
12  // quais a função isEven devolveu true, assim:
13  const evenNumbers = numbers.filter(isEven);
14  // mostrando na console o conteúdo do array
15  console.log(evenNumbers);
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ED-SEXTA\javaScript\tempCodeRunner
[
  2,  4,  6, 8,
 10, 12, 14
]
```

Usando o método reduce

- Temos também o método reduce, que recebe uma função com os seguintes parâmetros: **previousValue**, **currentValue**, **index** e **array**.
- Os parâmetros index e array são opcionais.
- Podemos usar essa função para **devolver** um valor que será somado a um **acumulador**, o qual será devolvido depois que o método **reduce** parar de executar.
- Isso pode ser muito útil se quisermos somar todos os valores de um array.
- Exemplo:

```
const soma = [1, 2, 3, 4, 5].reduce((a, b) => a + b);

console.log(soma);
```

- Na primeira vez que passar no loop o ***a*** valerá 0 e o ***b*** valerá 1, na segunda vez o ***a*** valerá 1 e o ***b*** valerá 2, na terceira vez o ***a*** valerá 3 e o ***b*** 3, na quarta vez ***a*** será igual a 6 e ***b*** igual a 4, na última vez ***a*** será 10 e ***b*** 5, o retorno então será 15.

Usando o método reduce

- Podemos usar essa função para **devolver** um valor que será somado a um **acumulador**, o qual será devolvido depois que o método **reduce** parar de executar.
- Isso pode ser muito útil se quisermos somar todos os valores de um array.
- Veja o exemplo:

```
numbers.reduce( (previous, current) => previous + current );
```

A saída será igual a 120.

O método ***reduce()*** reduz o array para um único valor.

```
1   let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
2   // A saída será igual a 120 que representa o somatório dos elementos
3   console.log (numbers.reduce((previous, current) => previous + current) );
4
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ED-SEXTA\javaScript\tempCodeRunnerFile.js"
120
```


Fonte: Loiane Groner, 2019. Pág. 94

ECMAScript 6 e as novas funcionalidades de array

Método	Descrição
@@iterator	Devolve um objeto iterator que contém os pares chave/valor do array; pode ser chamado sincronamente para obter a chave/valor dos elementos do array.
copyWithin	Copia uma sequência de valores do array na posição de um índice de início.
Entries	Devolve @@iterator, que contém pares chave/valor.
Includes	Devolve true caso um elemento seja encontrado no array, e false caso contrário. Foi adicionado na ES2016.
Find	Busca um elemento no array, dada uma condição desejada (função de call-back), e devolve o elemento caso seja encontrado.
findIndex	Busca um elemento no array, dada uma condição desejada (função de call-back), e devolve o índice do elemento caso seja encontrado.
Fill	Preenche o array com um valor estático.
From	Cria um novo array a partir de um array existente.
of	Cria um novo array a partir dos argumentos passados para o método.
values	Devolve @@iterator, contendo os valores do arrau.

Fonte: Loiane Groner, 2019. Pág. 94

Iterando com o laço for...of

- Junto com esses métodos, a API de Array também provê uma forma de iterar pelo array com o objeto **Iterator**, que pode ser obtido da instância do array e usado no laço **for...of**
- Vimos que podemos iterar por um array usando o laço **for** e o método **forEach**.
- A ES2015 (ES6) introduziu o laço **for...of** para iterar pelos valores de um array.
- Exemplo:

```
2   let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
3
4   for (const n of numbers) {
5       console.log(n % 2 === 0 ? n + " é par" : n + " é impar");
6   }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ED-SEXTA\javaScript\tempCodeRunner
1 é impar
2 é par
3 é impar
4 é par
5 é impar
6 é par
7 é impar
8 é par
9 é impar
10 é par
11 é impar
12 é par
13 é impar
14 é par
15 é impar
```

Usando o objeto @@iterator

- A classe Array também tem uma propriedade chamada **@@iterator**, introduzida na ES2015 (ES6 ou ES2015).
- Para usá-la, é necessário a propriedade **Symbol.iterator** do array.
- Então, podemos chamar individualmente o método **next( )** do **iterador** para obter o próximo valor.

```
1
2   // Usaremos um array co os valores de 1 a 15,
3   let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
4
5   let iterator = numbers[Symbol.iterator]();
6   console.log(iterator.next().value);  //1
7   console.log(iterator.next().value);  //2
8   console.log(iterator.next().value);  //3
9   console.log(iterator.next().value);  //4
10  console.log(iterator.next().value);  //4
11
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ED-SEXTA\javaScript\tempCodeRunnerF
1
2
3
4
5
```

Fonte: Loiane Groner, 2019. Pág. 945

Usando o objeto @@iterator

- Podemos apresentar todos os 15 valores do array **numbers** usando o seguinte código:

- Quando fizemos a iteração pelo array e não houver mais valores para iterar, o código **iterator.next( )** devolverá **undefined**.

```
1   // Usaremos um array co os valores de 1 a 15,
2   let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
3
4   let iterator = numbers[Symbol.iterator]();
5   //apresentando todos os 15 valores do array
6   for (const n of iterator) {
7     console.log(n);
8   }
9
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ED-SEXTA\javaScript\tempCodeRunner
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

Métodos entries, keys e values de array

- O método entries (ES2015) devolve @@iterator, que contém pares chave/valor.

- Como o array **numbers** contém somente números, **key** será a posição do array e **value** será o valor armazenado no índice do array.

```
1   // Usaremos um array com os valores de 1 a 15,
2   let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
3
4   let aEntries = numbers.entries();   // obtém um iterador de chave/valor
5   console.log(aEntries.next().value); // [0, 1] - posição 0, valor 1
6   console.log(aEntries.next().value); // [1, 2] - posição 1, valor 2
7   console.log(aEntries.next().value); // [2, 3] - posição 2, valor 3
8
9
```

```
PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ED-SEXTA\javaScript\tempCodeRunnerFile.
[ 0, 1 ]
[ 1, 2 ]
[ 2, 3 ]
```


Fonte: Loiane Groner, 2019. Pág. 96

Métodos entries, keys e values de array

- Também podemos usar o código a seguir como uma alternativa ao código anterior:

- Ser capaz de obter chave/valor será muito conveniente quando estivermos trabalhando com **conjuntos**, **dicionários** e **mapas hash** (hash maps).

```
1  // Usaremos um array com os valores de 1 a 15,
2  let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
3  // obtém um iterador de chave/valor
4  let aEntries = numbers.entries();
5
6  for (const n of aEntries) {
7    console.log(n);
8  }
9
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ED-SEXTA\javaScript\tempCodeRunner
[ 0, 1 ]
[ 1, 2 ]
[ 2, 3 ]
[ 3, 4 ]
[ 4, 5 ]
[ 5, 6 ]
[ 6, 7 ]
[ 7, 8 ]
[ 8, 9 ]
[ 9, 10 ]
[ 10, 11 ]
[ 11, 12 ]
[ 12, 13 ]
[ 13, 14 ]
[ 14, 15 ]
```

Métodos entries, keys e values de array

- O método **keys( )** devolve **@@iterator**, que contém as chaves do array.
- Para o array **numbers, keys( )** conterá os índices do array.
- Quando não houver mais valores para iterar, o código **aKeys.next( )** devolverá **undefined** como value e **done** como **true**.
- Se **done** tiver um valor igual a **false**, isso significa que ainda há mais chaves para iterar no array.

```
1   // Usaremos um array com os valores de 1 a 15,
2   let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
3
4   const aKeys = numbers.keys(); // obtém um iterator de chaves
5   console.log(aKeys.next()); // {value: 0, done: false}
6   console.log(aKeys.next()); // {value: 1, done: false}
7   console.log(aKeys.next()); // {value: 2, done: false}
8
9
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ED-SEXTA\javaScript\tempCodeRunnerF
{ value: 0, done: false }
{ value: 1, done: false }
{ value: 2, done: false }
```


Fonte: Loiane Groner, 2019. Pág. 96

Métodos entries, keys e values de array

- O método **values( )** devolve **@@iterator**, que contém os **valores** do array.

```
1   // Usaremos um array com os valores de 1 a 15,
2   let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
3
4   const aValues = numbers.values(); // obtém um iterator de valores
5   console.log(aValues.next()); // {value: 1, done: false}
6   console.log(aValues.next()); // {value: 2, done: false}
7   console.log(aValues.next()); // {value: 3, done: false}
8
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ED-SEXTA\javaScript\tempCodeRunner
{ value: 1, done: false }
{ value: 2, done: false }
{ value: 3, done: false }
```


Fonte: Loiane Groner, 2019. Pág. 96

Ordenando elementos

- O JavaScript também tem um método de **ordenação** e dois métodos de **pesquisa** disponíveis.
- Em primeiro lugar, vamos usar nosso array **numbers** e deixar os elementos fora de ordem (invertido) com o método **reverse( )**.
- Em seguida, podemos aplicar o método **sort( )**:

```
1   // Usaremos um array com os valores de 1 a 15,
2   let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
3   // o último item será o primeiro e vice-versa
4   numbers.reverse();
5   console.log('Array numbers com reverse: ' + numbers.join(', '));
6   // o método sort deixa os elementos em ordem lexicográfica(alfabética)
7   // e pressupõe que todos os elementos são strings
8   numbers.sort();
9   console.log('Array numbers com sort: ' + numbers.join(', '));
10
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ED-SEXTA\javaScript\tempCodeRunnerFile.js"
Array numbers com reverse: 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
Array numbers com sort: 1, 10, 11, 12, 13, 14, 15, 2, 3, 4, 5, 6, 7, 8, 9
```

Ordenando elementos

- Em seguida, podemos aplicar o método sort( ).
- No entanto ele não está ordenado corretamente porquê o método sort deixa os elementos em ordem lexicográfica (alfabética) e pressupões que todos os elementos são strings.
- Podemos também implementar nossa própria função de comparação:

```
1   // Usaremos um array com os valores de 1 a 15,
2   let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
3   // o último item será o primeiro e vice-versa
4   numbers.reverse();
5   console.log('Array numbers com reverse: ' + numbers.join(', '));
6   // o método sort deixa os elementos em ordem lexicográfica(alfabética)
7   // e pressupõe que todos os elementos são strings
8   numbers.sort();
9   console.log('Array numbers com sort: ' + numbers.join(', '));
10
11  function compare(a, b) {
12      if (a < b) {
13          return -1;
14      }
15      if (a > b) {
16          return 1;
17      }
18      // a deve ser igual a b
19      return 0;
20  }
21  numbers.sort(compare);
22  console.log('Array numbers executando a função compare: ' + numbers.join(', '));
23
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ED-SEXTA\javaScript\tempCodeRunnerFile.js"
Array numbers com reverse: 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
Array numbers com sort: 1, 10, 11, 12, 13, 14, 15, 2, 3, 4, 5, 6, 7, 8, 9
Array numbers executando a função compare: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
```

Ordenando strings

- Ordenando um array de strings com o método **sort( )**

- O método **sort( )** deixa os elementos em ordem lexicográfia (alfabética) e pressupõe que todos os elementos são strings.

```
1
2   // ordenando um array de strings alfabeticamente
3   let names = ['Maria', 'Angelica', 'Luiza', 'Kelly', 'Rafaela', 'Marcela', 'Beatriz']
4
5   console.log(names.sort().join(', '));
6
7
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ED-SEXTA\javaScript\tempCodeRunnerFile.js"
Angelica, Beatriz, Kelly, Luiza, Marcela, Maria, Rafaela
```


Fonte: Loiane Groner, 2019. Pág. 101

Métodos de Pesquisa

- O método **indexOf( )**, que devolve o índice do primeiro elemento correspondente ao argumento passado.
- O método **lastIndexOf( ),** que devolve o índice do último elemento encontrado, correspondente ao argumento passado.
- Caso o valor não exista no array, o método devolve **-1**.

```
1   // Usaremos um array com os valores de 1 a 15,
2   let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
3   // indexOf devolve o índice do primeiro elemento correspondente
4   console.log('O indice do primeiro elemento 10 é: ' + numbers.indexOf(10));
5   console.log('O indice do elemento 100 que não existe no array é: ' + numbers.indexOf(100));
6   //vamos adicionar mais um elemento 10 ao final do array
7   numbers.push(10);
8   // mostrando o array numbers com mais um elemento adicionado
9   console.log('Mostrando novamente o array numbers com mais um elemento adicionado: ');
10  console.log(numbers.join(', '));
11  // lastIndexOf devolve o índice do último elemento correspondente
12  console.log('O indice do último elemento 10 é: ' + numbers.lastIndexOf(10));
13  console.log('O indice do último elemento 100 que não existe no array é: ' + numbers.lastIndexOf(100));
14
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE Code

```
[Running] node "c:\Users\admin\Desktop\ED-SEXTA\javaScript\tempCodeRunnerFile.js"
O indice do primeiro elemento 10 é: 9
O indice do elemento 100 que não existe no array é: -1
Mostrando novamente o array numbers com mais um elemento adicionado:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 10
O indice do último elemento 10 é: 15
O indice do último elemento 100 que não existe no array é: -1
```

ECMAScript 2015 – os métodos find e findIndex

- Os métodos **find** e **findIndex** recebem uma função de callback, a qual buscará um valor que satisfaça a condição presente na função de teste (call-back).

- A diferença entre **find** e **findIndex** é que o método **find** devolve o primeiro valor do array que satisfaça a condição proposta.

- O método findIndex, por outro lado, devolve o índice do primeiro valor do array que satisfaça a condição.

- Neste exemplo estamos verificando se o array **number** contém algum múltiplo de 13:

- Caso o valor não seja encontrado, **undefined** será devolvido.

```
1   // Usaremos um array com os valores de 1 a 15,
2   let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
3
4   function multipleOf13(element, index, array) {
5       return (element % 13 == 0);
6   }
7   console.log(numbers.find(multipleOf13));
8   console.log(numbers.findIndex(multipleOf13));
9
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ED-SEXTA\javaScript\tempCodeRunner
13
12
```

ECMAScript 2015 – usando o método includes

- Os método includes devolve true caso um elemento seja encontrado no array, e false caso contrário.
- Nesse exemplo, **includes**(15) devolverá **true** e **includes**(20) devolverá **false**, pois o elemento 20 não existe no array **numbers**.

```
1   // Usaremos um array com os valores de 1 a 15,
2   let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
3
4   console.log(numbers.includes(15));
5   console.log(numbers.includes(20));
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ED-SEXTA\javaScript\tempCodeRunner
true
false
```

- Também é possível passar um índice de início a partir do qual queremos que o **array** faça a pesquisa do valor:

```
8   let numbers2 = [7, 6, 5, 4, 3, 2, 1];
9   console.log(numbers2.includes(4,5));
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ED-SEXT
false
```

- A saída do exemplo acima será **false** porque o elemento **4** não existe após a posição **5**.

Fonte: Loiane Groner, 2019. Pág. 103

Convertendo um array em uma string

- Se quisermos exibir todos os elementos do **array** em uma única **string**, podemos usar o método **toString**:

```
1   // Usaremos um array com os valores de 1 a 15,
2   let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
3
4   console.log(numbers.toString());
5
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ED-SEXTA\javaScript\tempCodeRunner
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
```

- Se quisermos separar os elementos com um separador diferente, o método **join** poderá ser usado:

```
1   // Usaremos um array com os valores de 1 a 15,
2   let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];
3
4   console.log(numbers.join('-'));
5
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ED-SEXTA\javaScript\tempCodeRunner
1-2-3-4-5-6-7-8-9-10-11-12-13-14-15
```

Fonte: Loiane Groner, 2019. Pág. 104

Mais recursos de Array JavaScript

Existem alguns recursos ótimos que você poderá usar para ampliar o seu conhecimento sobre arrays e seus métodos:

- O **Mozilla** também tem uma página excelente sobre **arrays** e seus métodos, com ótimos exemplos em https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array

- A biblioteca **Lo-Dash** e igualmente muito útil quando trabalhamos com **arrays** em projetos **JavaScript**: https://lodash.com

Fonte: Loiane Groner, 2019. Pág. 104