Lógica computacional

JavaScript

Array

Arrays são objetos semelhantes a listas que vêm com uma série de métodos embutidos para realizar operações.

Nem o tamanho de um array JavaScript nem os tipos de elementos são fixos.

Já que o tamanho de um array pode ser alterado a qualquer momento e os dados podem ser armazenados em posições não contíguas.

Criando um Array

```
var frutas = ['Maçã', 'Banana'];


console.log(frutas.length);
// 2
```

Acessar um item (index) do Array

```
var primeiro = frutas[0];

// Maçã


var ultimo = frutas[frutas.length - 1];

// Banana
```

Iterar um Array

```
frutas.forEach(function (item, indice, array) {
  console.log(item, indice);
});
// Maçã 0
// Banana 1
```

Adicionar um item ao final do Array

```
var adicionar = frutas.push('Laranja');

// ['Maçã', 'Banana', 'Laranja']
```

Remover um item do final do Array

```
var ultimo = frutas.pop(); // remove Laranja (do final)

// ['Maçã', 'Banana'];
```

Remover do início do Array

```
var primeiro = frutas.shift(); // remove Maçã do início

// ['Banana'];
```

Adicionar ao início do Array

```
var adicionar = frutas.unshift('Morango') // adiciona ao início

// ['Morango', 'Banana'];
```

Procurar o índice de um item na Array

```
frutas.push('Manga');

// ['Morango', 'Banana', 'Manga']


var pos = frutas.indexOf('Banana');

// 1
```

Remover um item pela posição do índice

```
var removedItem = frutas.splice(pos, 1); // é assim que se remove um item

// ['Morango', 'Manga']
```

Remover itens de uma posição de índice

var vegetais = ['Repolho', 'Nabo', 'Rabanete', 'Cenoura'];

console.log(vegetais); // ['Repolho', 'Nabo', 'Rabanete', 'Cenoura']

// Isso é como se faz para remover itens, n define o número de itens a se remover,

var itensRemovidos = vegetais.splice(1, 2); // a partir da posição (pos) em direção ao fim da array.

console.log(vegetais); // ['Repolho', 'Cenoura'] (o array original é alterado)

console.log(itensRemovidos); // ['Nabo', 'Rabanete']

Copiar um Array

```
var copiar = frutas.slice(); // é assim que se copia

// ['Morango', 'Manga']
```

Criando um array bi-dimensional

O exemplo a seguir cria um tabuleiro de xadrez usando dois arrays bi-dimensionais de string. A primeira jogada é feita copiando o 'p' em 6,4 para 4,4. A posição antiga de 6,4 é colocada em branco.

```
var board =
[ ['R','N','B','Q','K','B','N','R'],
  ['P','P','P','P','P','P','P','P'],
  [' ',' ',' ',' ',' ',' ',' ',' '],
  [' ',' ',' ',' ',' ',' ',' ',' '],
  [' ',' ',' ',' ',' ',' ',' ',' '],
  [' ',' ',' ',' ',' ',' ',' ',' '],
  ['p','p','p','p','p','p','p','p'],
  ['r','n','b','q','k','b','n','r']];
console.log(board.join('\n') + '\n\n');

// Fazendo o King's Pawn avançar 2
board[4][4] = board[6][4];
board[6][4] = ' ';
console.log(board.join('\n'));
```

Saída:

```
R,N,B,Q,K,B,N,R
P,P,P,P,P,P,P,P
 , , , , , , ,
 , , , , , , ,
 , , , , , , ,
 , , , , , , ,
p,p,p,p,p,p,p,p
r,n,b,q,k,b,n,r

R,N,B,Q,K,B,N,R
P,P,P,P,P,P,P,P
 , , , , , , ,
 , , , , , , ,
 , , , ,p, , ,
 , , , , , , ,
p,p,p,p, ,p,p,p
r,n,b,q,k,b,n,r
```

Utilizando um array para tabular um conjunto de valores

```
values = [];
for (var x = 0; x < 10; x++){
 values.push([
  2 ** x,
  2 * x ** 2
 ])
};
console.table(values)
```

Saída:

```
0  1  0
1  2  2
2  4  8
3  8  18
4  16  32
5  32  50
6  64  72
7  128  98
8  256  128
9  512  162
```

Array.prototype.filter()

O método filter() cria um novo array com todos os elementos que passaram no teste implementado pela função fornecida.

Sintaxe

var newArray = arr.filter(callback[, thisArg])

```
function isBigEnough(value) {
  return value >= 10;
}

var filtered = [12, 5, 8, 130, 44].filter(isBigEnough);
// filtrado é [12, 130, 44]
```

Array.prototype.find()

O método find() retorna o valor do primeiro elemento do array que satisfizer a função de teste provida. Caso contrario, undefined é retornado.

Sintaxe

arr.find(callback(element[, index[, array]])[, thisArg])

```
const array1 = [5, 12, 8, 130, 44];

const found = array1.find(element => element > 10);

console.log(found);
```

Array.prototype.includes()

O método includes() determina se um array contém um determinado elemento, retornando true ou false apropriadamente.

Sintaxe

array.includes(searchElement[, fromIndex])

```
[1, 2, 3].includes(2);     // true

[1, 2, 3].includes(4);     // false

[1, 2, 3].includes(3, 3);  // false

[1, 2, 3].includes(3, -1); // true

[1, 2, NaN].includes(NaN); // true
```

Array.prototype.map()

O método map() invoca a função callback passada por argumento para cada elemento do Array e devolve um novo Array como resultado.

Sintaxe

arr.map(callback[, thisArg])

```
var numbers = [1, 4, 9];
var roots = numbers.map(Math.sqrt);
// roots é [1, 2, 3], numbers ainda é [1, 4, 9]

var numbers = [1, 4, 9];
var doubles = numbers.map(function(num) {
  return num * 2;
});
// doubles é agora [2, 8, 18]. numbers ainda é [1, 4, 9]
```

Array.prototype.reduce()

O método reduce() executa uma função reducer (fornecida por você) para cada elemento do array, resultando num único valor de retorno.

Sintaxe

array.reduce(callback(acumulador, valorAtual[, index[, array]])[, valorInicial]))

```
const array1 = [1, 2, 3, 4];

// 0 + 1 + 2 + 3 + 4
const initialValue = 0;
const sumWithInitial = array1.reduce(
  (previousValue, currentValue) => previousValue + currentValue,
  initialValue
);

console.log(sumWithInitial);
// expected output: 10
```