\*Callback

1. forEach()

Hàm này sẽ giúp lặp qua các phần tử của mảng

Cú pháp

array.forEach(function(currentValue, index, arr), thisValue)

Trong đó:

* currentValue - giá trị của phần tử hiện tại.
* index - chỉ số của phần tử hiện tại.
* arr - mảng mà phần tử hiện tại thuộc về.
* thisValue - tham số không bắt buộc. Nếu được truyền vào thì thisValue sẽ được sử dụng làm giá trị this. Nếu không được truyền vào thì giá trị "this" là "undefined".

VD:

let ranks = ['A', 'B', 'C'];

ranks.forEach(function (e) {

    console.log(e); // A B C

});

1. map()

Hàm này có công dụng tương tự như vòng lăp forEach. Hàm map() sẽ lặp qua từng phần tử của mảng

Cú pháp

array.map(function(currentValue, index, arr), thisValue)

Trong đó:

* currentValue - giá trị của phần tử hiện tại.
* index - chỉ số của phần tử hiện tại.
* arr - mảng mà phần tử hiện tại thuộc về.
* thisValue - tham số không bắt buộc. Nếu được truyền vào thì thisValue sẽ được sử dụng làm giá trị this. Nếu không được truyền vào thì giá trị "this" là "undefined".

Giá trị trả về của hàm map trong js là một mảng mới, với số lượng phần tử bằng với mảng cũ, nhưng giá trị của các phần tử thì được quyết định bởi lệnh return của hàm map.

VD:

var numbers = [5, 10, 20, 22, 60];

var new\_number = numbers.map(function(item){

    return item \* 2; // nhân đôi giá trị

});

console.log(new\_number);

// Kết quả: [10, 20, 40, 44, 120]

**3.fillter**

1. Reduce

Hàm này sẽ duyệt qua từng phần tử trong mảng, trả về một giá trị duy nhất là kết quả tích lũy của hàm

Cú pháp

array.reduce(function(total, currentValue, currentIndex, arr), initialValue)

Trong đó:

* total - giá trị trả lại trước đó của function, chính là giá trị của lệnh return cho mỗi lần lặp.
* currentValue - giá trị của phần tử hiện tại.
* currentIndex- chỉ số của phần tử hiện tại.
* arr - mảng mà phần tử hiện tại thuộc về.
* initialValue - tham số không bắt buộc. Nếu được truyền vào thì initialValue sẽ được sử dụng làm giá trị ban đầu, còn không thì nó sẽ lấy giá trị của phần tử đầu tiên

1. some()

Hàm này kiểm tra xem có ít nhất phần tử nào trong array vượt qua được điều kiện đưa ra không. Kết quả trả về boolean

Cú pháp

array.some(function(currentValue, index, arr), thisValue)

Trong đó:

* currentValue - giá trị của phần tử hiện tại.
* index - chỉ số của phần tử hiện tại.
* arr - mảng mà phần tử hiện tại thuộc về.
* thisValue - tham số không bắt buộc. Nếu được truyền vào thì thisValue sẽ được sử dụng làm giá trị this. Nếu không được truyền vào thì giá trị "this" là "undefined".

VD:

const arr = [3, 9, 7, 6];

// thằng nào thi < 5 trượt

const idiot = arr.some(num => num < 5);

console.log(idiot); // output: true // có thằng trượt :D

1. every()

Hàm này ngược lại some()  kiểm tra xem tất cả các phần tử trong array vượt qua được điều kiện đưa ra không. Kết quả trả về boolean

Cú pháp

array.every(function(currentValue, index, arr), thisValue)

Trong đó:

* currentValue - giá trị của phần tử hiện tại.
* index - chỉ số của phần tử hiện tại.
* arr - mảng mà phần tử hiện tại thuộc về.
* thisValue - tham số không bắt buộc. Nếu được truyền vào thì thisValue sẽ được sử dụng làm giá trị this. Nếu không được truyền vào thì giá trị "this" là "undefined".

VD:

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

// check tất cả có > 4 không

const greaterFour = arr.every(num => num > 4);

console.log(greaterFour); // output: false sai rồi

1. sort()

sắp xếp

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

const alpha = ['e', 'a', 'c', 'u', 'y'];

// sắp xếp theo thứ tự giảm dần

descOrder = arr.sort((a, b) => a > b ? -1 : 1);

console.log(descOrder); // output: [6, 5, 4, 3, 2, 1]

// sắp xếp theo thứ tự tăng dần

ascOrder = alpha.sort((a, b) => a > b ? 1 : -1);

console.log(ascOrder); // output: ['a', 'c', 'e', 'u', 'y']

1. from()

Hàm này cho phép tạo các array từ một kiểu dữ liệu khác,

VD:

const name = 'javascript';

const nameArray = Array.from(name);

console.log(name); // output: javascript

console.log(nameArray); // output: ['j', 'a', 'v', 'a', 's', 'c', 'r', 'i', 'p', 't']