**Đặt vấn đề**

Cho một mảng [numbers](https://completejavascript.com/values-types-va-operators-trong-javascript-phan-1/) = [1, 2, 3, 4, 5, 6,7, 8, 9, 10]. Hãy tính tổng các số trong mảng và hiển thị ra console.

Thông thường, chúng ta sẽ giải bài toán này như sau:

const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

let sum = 0;

for(let i = 0; i < numbers.length; i++) {

sum += numbers[i];

}

console.log(sum);

// => 55

Cách giải trên là khá dễ hiểu. Chúng ta lấy từng phần tử trong mảng ra theo chỉ số, rồi cộng với biến *sum*.

Tuy nhiên có một vấn đề với cách này (cũng không hẳn là một vấn đề), đó là việc sử dụng chỉ số **i**có thể khiến bạn bị nhầm với một biến số trước đó. Dẫn đến những sai lầm không thể đoán trước.

Lúc này, **forEach** JavaScript xuất hiện giúp bạn giải quyết vấn đề trên.

**Cách triển khai hàm forEach trong JavaScript**

Một cách đơn giản để triển khai hàm forEach như sau:

function forEach(array, action){

for(let i = 0; i < array.length; i++)

action(array[i]);

}

Hàm *forEach* trên nhận đầu vào là một mảng *array* và một hàm số *action* – thực hiện hành động đối với mỗi phần tử của mảng (*array[i]*).

**Áp dụng hàm forEach để giải quyết bài toán trên**

const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

let sum = 0;

forEach(numbers, function(element){

sum += element;

});

console.log(sum);

// => 55

Ta có thể diễn giải thuật toán trên bằng lới như sau: *Với mỗi phần tử trong mảng numbers, ta sẽ lấy nó ra và cộng dồn với biến sum. Kết quả thu được, sẽ ghi ra console.*

Nếu so sánh với cách trình bày ở trên thì cách này rõ ràng sẽ dài hơn. Tuy nhiên, đây chỉ là một ví dụ cơ bản, nên ta sẽ không bàn về độ dài, ngắn của code. Vấn đề mình và bạn quan tâm trước tiên đó là: **code-đọc-dễ-hiểu**.

Theo quan điểm cá nhân mình thì việc sử dụng hàm **forEach** giúp code gần giống với ngôn ngữ tự nhiên hơn. Không biết quan điểm của bạn về vấn đề này thế nào?

**Giới thiệu hàm forEach trong JavaScript**

[JavaScript](https://completejavascript.com/category/javascript-co-ban/) cung cấp sẵn phương thức **forEach**– là một phương thức có sẵn của [array](https://completejavascript.com/array-sorting-van-de-muon-thuo/), để thực hiện hành động với mỗi phần tử của mảng.

**Cú pháp**

arr.forEach(function callback(currentValue, index, array) {

// your iterator

}[, thisArg]);

**Giải thích**

**callback**: là hàm để thực hiện với mỗi phần tử của mảng, bao gồm 3 tham số:

* **currentValue**: phần tử hiện tại đang được xử lý của array.
* **index**: chỉ số của phần tử hiện tại đang được xử lý của array.
* **array**: mảng hiện tại đang gọi hàm forEach.

**thisArg**: giá trị được sử dụng như là **this**, là tham chiếu tới đối tượng khi thực hiện hàm callback.

**Chú ý:**Nếu *thisArg* bị bỏ qua thì mặc định giá trị đó là **undefined**.

**Ví dụ in ra nội dung của mảng**

function printContentArray(array){

array.forEach(function print(element){

console.log(element);

});

}

printContentArray([1, 3, 5]);

/\*

\* 1

\* 3

\* 5

\*/

**Ví dụ sử dụng *thisArg***

function Counter() {

this.sum = 0;

this.count = 0;

}

Counter.prototype.add = function(array) {

array.forEach(function(entry) {

this.sum += entry;

++this.count;

}, this);

// ^---- Note

};

const obj = new Counter();

obj.add([2, 5, 9]);

console.log(obj.count); // => 3

console.log(obj.sum); // => 16

**Ưu, nhược điểm của việc sử dụng forEach**

Bất cứ thứ gì cũng có hai mặt của nó, **forEach** JavaScript cũng không ngoại lệ. Sau đây là một số ưu, nhược điểm của nó:

**Ưu điểm**

* Code rõ ràng, gần với ngôn ngữ tự nhiên
* Trong nhiều trường hợp sẽ ngắn gọn hơn việc sử dụng *for*, *while*.

**Nhược điểm**

* Chắc chắn code sẽ chạy chậm hơn so với việc sử dụng vòng lặp (tuy nhiên không đáng kể)
* Không giống các ngôn ngữ lập trình khác như C/C++, Java (hầu như chỉ sử dụng vòng lặp)

Trên đây là một số ưu, nhược điểm của việc sử dụng forEach. Mình không khuyên bạn nên sử dụng phương thức này. Đây chỉ đơn giản là một cách thức duyệt mảng, bạn có thể sử dụng nó hoặc bạn chỉ cần sử dụng vòng lặp for, while.

*It’s the choice.*

**Từ JavaScript forEach đến các phương thức khác**

Ngoài forEach, JavaScript còn cung cấp một số phương thức khác, tương tự dành cho array. Đó là: *entries, every, filter, find, findIndex, keys, map, reduce, reduceRight, some, values.*

[**array.prototype.entries()**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/entries)

Trả về một mảng đối tượng mới, chứa **key/value** cho mỗi phần tử trong array.

a.entries()

Để duyệt mảng đối tượng mới này, ta sử dụng vòng lặp **for…of**

const a = ['a', 'b', 'c'];

const iterator = a.entries();

for (let e of iterator) {

console.log(e);

}

/\*

\* [0, 'a']

\* [1, 'b']

\* [2, 'c']

\*/

[**array.prototype.every()**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/every)

Trả về true nếu như tất cả các phần tử trong mảng thoả mãn 1 hàm kiểm tra, ngược lại trả về false.

arr.every(callback[, thisArg])

Tham số truyền vào giống hệt với hàm forEach.

Ví dụ sau trả về true nếu tất cả các phần tử trong mảng thoả mãn đều >= 10:

function isBigEnough(element, index, array) {

return element >= 10;

}

console.log([12, 5, 8, 130, 44].every(isBigEnough)); // => false

console.log([12, 54, 18, 130, 44].every(isBigEnough)); // => true

[**array.prototype.filter()**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter)

Giống như cái tên của nó, phương thức này dùng để lọc ra các phần tử trong mảng thoả mãn một điều kiện cho trước. Và trả về một mảng mới chứa những phần tử đó.

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

Tham số truyền vào giống hệt với hàm forEach.

Ví dụ sau lọc ra những phần tử trong mảng có giá trị >= 10:

function isBigEnough(value) {

return value >= 10;

}

const filtered = [12, 5, 8, 130, 44].filter(isBigEnough);

console.log(filtered);

// [12, 130, 44]

[**array.prototype.find()**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find)

Trả về giá trị của phần tử đầu tiên trong mảng thoả mãn điều kiện cho trước, nếu không tìm thấy thì trả về **undefined**.

var var1 = arr.find(callback[, thisArg])

Tham số truyền vào cũng giống hệt như hàm forEach.

Ví dụ sau tìm ra một [object](https://completejavascript.com/javascript-object-last-but-not-least/) với tên của nó thoả mãn điều kiện cho trước:

const inventory = [

{name: 'apples', from: 'usa'},

{name: 'bananas', from: 'vietnam'},

{name: 'cherries', from: 'usa'},

{name: 'cherries', from: 'china'}

];

function findCherries(fruit) {

return fruit.name === 'cherries';

}

console.log(inventory.find(findCherries));

// {name: 'cherries', from: 'usa'}

[**array.prototype.findIndex()**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/findIndex)

Trả về chỉ số đầu tiên của phần tử thoả mãn điều kiện cho trước, ngược lại trả về -1.

var id = arr.findIndex(callback[, thisArg])

Tham số truyền vào cũng giống hệt như hàm forEach.

Ví dụ sau tìm ra chỉ số object với tên của nó thoả mãn điều kiện cho trước:

const inventory = [

{name: 'apples', from: 'usa'},

{name: 'bananas', from: 'vietnam'},

{name: 'cherries', from: 'usa'},

{name: 'cherries', from: 'china'}

];

function findCherries(fruit) {

return fruit.name === 'cherries';

}

console.log(inventory.findIndex(findCherries));

// => 2

[**array.prototype.map()**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map)

Trả về mảng mới với mỗi phần tử là kết quả của việc gọi hàm callback ứng với mỗi phần tử của mảng ban đầu.

var new\_array = arr.map(callback[, thisArg])

Tham số truyền vào cũng giống hệt như hàm forEach trong JavaScript.

Ví dụ sau trả về mảng mới mà mỗi phần tử của mảng mới là căn bậc hai của phần tử tương ứng trong mảng ban đầu:

const numbers = [1, 4, 9];

const roots = numbers.map(Math.sqrt);

console.log(roots);

// => [1, 2, 3]

console.log(numbers);

// => [1, 4, 9]

Những phương thức còn lại bạn có thể tham khảo thêm tại [đây](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array). Mình sẽ không trình bày thêm vì bài viết sẽ quá dài.

**Thực hành**

**1. Flattening**

Cho một mảng hai chiều. Hãy chuyển mảng đó thành mảng một chiều:

* Sử dụng phương thức [array.prototype.reduce](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce) và concat(). ([tham khảo](https://github.com/completejavascript/practical-javascript/blob/master/iteration/iteration-flattening-completejavascript.com.js))
* Sử dụng vòng lặp for, while thông thường. ([tham khảo](https://github.com/completejavascript/practical-javascript/blob/master/iteration/loop-flattening-completejavascript.com.js))

Ví dụ:

var arrays = [[1, 2, 3], [4, 5], [6]];

/\*

\* Your code here.

\* // => [1, 2, 3, 4, 5, 6]

\*/

**2. Every and some**

Hãy viết hàm every() và some() thoả mãn:

console.log(every([NaN, NaN, NaN], isNaN));

// => true

console.log(every([NaN, NaN, 4], isNaN));