**Javascript**

forEach(ts1, ts2)

Duyệt qua hết mảng và có 2 tham số truyền vào: 1 là chỉ số index, 2 là biến lưu các giá trị trong mảng.

every(ts1, ts2)

Dùng để kiểm tra một điều kiện nào đó hoàn toàn đúng với tất cả phần tử trong mảng (kiểu trả về là boolean)

filter( function(ts1, ts2) {

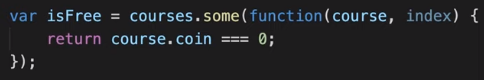
return Điều\_kiện;

}) ;

Dùng để lọc các phần tử thỏa mãn điều kiện nào đó. Trả về 1 mảng mới gồm các phần tử thỏa mãn điều kiện.

some()

Dùng để kiểm tra điều kiện nếu chỉ cần 1 phần tử đúng thì sẽ là đúng (kiểu trả về là boolean)



map(function(ts1, ts2, ts3) {

/\*

Ts1: 1 mảng mới được khởi tạo để truyền từ mảng củ sang mảng mới

Mảng này trả về cái gì thì sẽ lấy cái đó

Ts2: chỉ số index của mảng

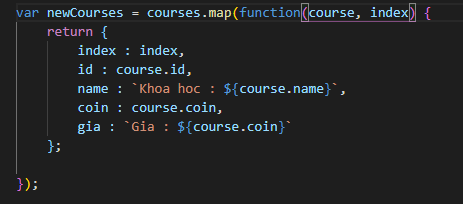
Ts3: originArray: trả về mảng củ trước đó (ít sử dụng)

\*/

});

Dùng để chỉnh sửa, thay đổi element của 1 mảng

* Trong map 1 là function
* Duyệt qua từng phần tử của mảng
* Lấy phần tử thứ 1 gọi ngược lại function truyền vào



reduce(ts1, ts2)

ts1: là 1 function (ts1, ts2, ts3, ts4) {

ts1: biến lưu trữ (accumulator)

ts2: giá trị hiện tại

ts3: index

ts4: originArray: trả về mảng củ trước đó (ít sử dụng)

}

ts2: giá trị khởi tạo (được gán cho biến lưu trữ phía trên), tùy vào bài toán: số, chữ, object,.. ( ~ var tong = 0;)

**for…of : lặp qua các phần tử trong 1 mảng**

**for(var** x **of** tên\_mảng**){}**

x: đại diện cho từng phần tử trong mảng.

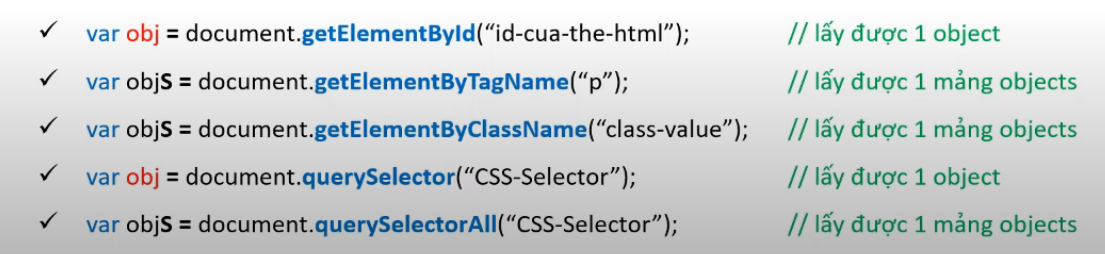
**for…in : lặp qua các key trong 1 object**

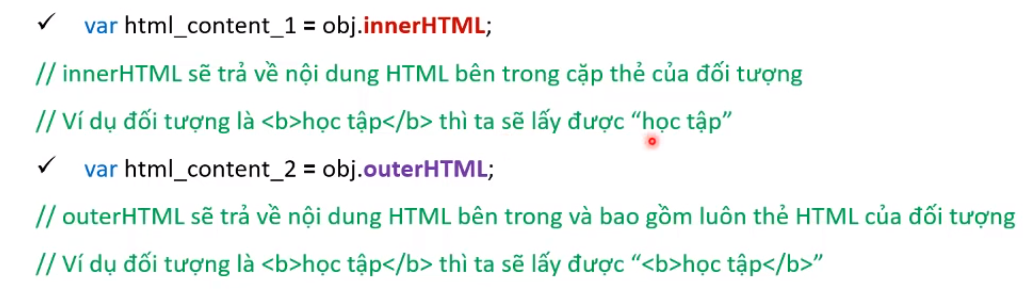
**for(var** x **in** tên\_mảng**){}**

x: đại diện cho các key trong mảng

\* Nếu muốn lấy value thông qua key thì tên\_mảng[key]

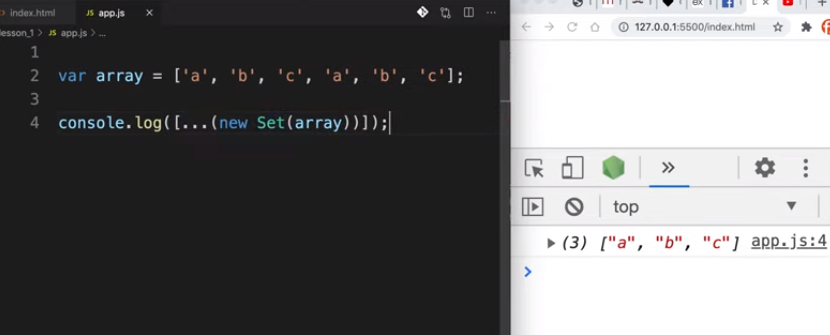
DOM





\*SET trong JS

Set() : các gía trị trong set sẽ không được lặp lại



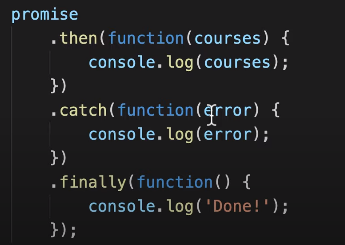
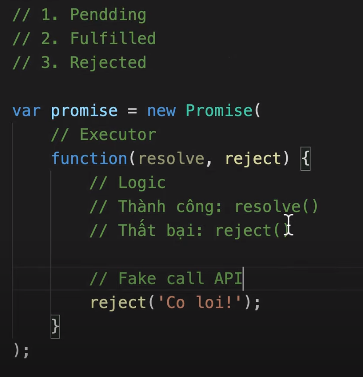
… Là toán tử giải như ở ví dụ trên ban đầu Set() sẽ trả về một object. Nhưng ta muốn kết quả là mảng thì trước tiên phải giải các phần tử trong object đó ra ngoài để cho nó thành phần tử tự do. Sau đó thì dùng ký hiệu của mảng [] bọc lại. Ta được kết quả

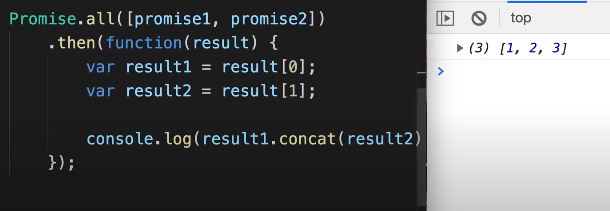
**PROMISE**

Promise: được sinh ra để xử lý quy tắc bất đồng bộ. Trước khi có promise thì chúng ta sử dụng callback mà callback thì xảy ra vấn đề đó là callback hell nó sẽ bị sâu vào rất khó nhìn, khó hiểu do đó promise được sinh ra.

Để tạo ra được promise thì chúng ta sử dụng từ khóa new với với promise và trong constructor của nó chúng ta sẽ truyền vào một executor function. Trong executor function sẽ nhận 2 tham số một là resolve 2 là reject. Resolve, chúng ta sẽ gọi nó khi thao tác xử lý logic thành công. Reject được gọi khi thao tác xử lý thất bại.

Khi sử dụng promise, chúng ta sẽ sử dụng phương thức .then() và .catch(). Then() sẽ nhận một callback function khi promise nhận resolve. Và vào catch khi promise nhận reject.





Promise.all: chạy vào then nếu 2 cái đều là resolve. Chỉ cần 1 thằng là reject thì nó sẽ hỏng.

**ES6**

1. **var, let, const**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Var | Let | Const |
| Truy cập từ ngoài khối code | Được phép | Không được phép | Không được phép |
| Hosting (tự động nhấc biến khai báo lên/ gán biến trước mới khai báo) | Được phép | Không được phép | Không được phép |
| Assignment (ví dụ: khai báo a = 1, sau đó gán lại a = 3)/ Không thể sử dụng toán tử gán lần 2 | Được phép | Được phép | Không được phép |

**Khi nào sử dụng:**

Code thuần: var

Sử dụng thư viện: const, let

Khi định nghĩa biến và không gán lại biến đó thì dùng const

Khi cần gán lại giát trị cho biến thì dùng let

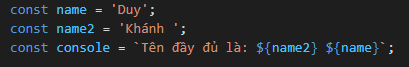
1. **Arrow function**

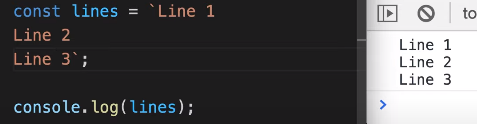
const a = (number) => { }

const sum = (a, b) => a + b;

Phía sau dấu => là một return

1. **Template literals (Template string)**



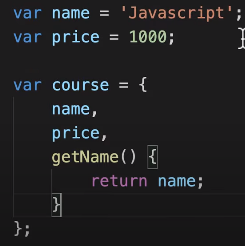


1. **Object literals**

Định nghĩa key: value cho object

Định nghĩa method cho object

Định nghĩa key cho object dưới dạng biến





1. **JSON**

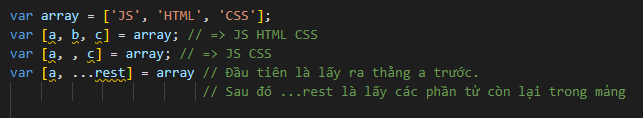
JSON: là một định dạng dữ liệu (chuỗi)

Stringify: từ javascript types -> JSON

Parse: từ JSON -> javascript types

1. **Destructuring (phân rã: array, object)**

Array



Object

