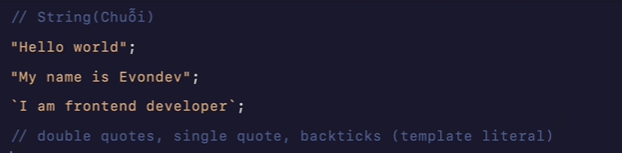
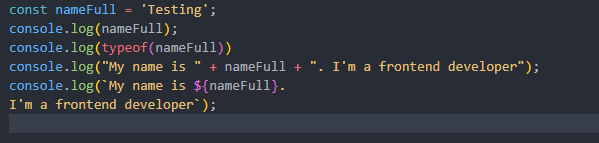
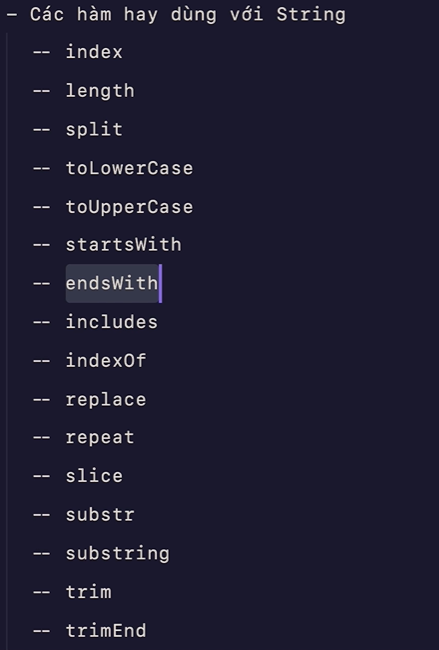
# String

## Giới thiệu

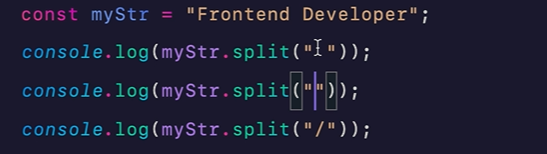




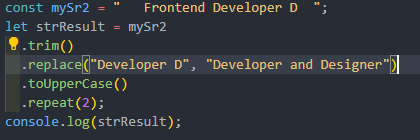
## Các phương thức ( method ) trong string



* Length
* Split( “ ”) :

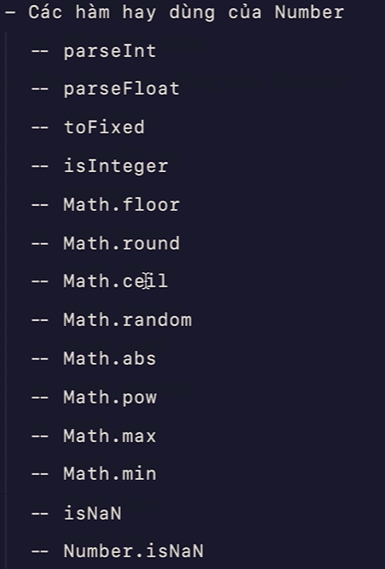


* toLowerCase()
* toUpperCase()
* startsWith('Testing J')
* endsWith('Testing J')
* includes('est')
* indexOf(‘D’)
* lastIndexOf(‘D’)
* replace('es','NB')
* replaceAll('es','NB')
* slice(start, end) // not include end char, nếu không có end là lấy hết chuỗi
* trim(), trimStart(), trimEnd() // Remove space at the begin end end, left, right
* charAt(5) // Get char at index number
* substr(2, 5)   // Get a subtring from index 2 with length is 5
* substring(2, 5))  // Get a substring from index 2 to index 5 - 1
* combine multi method:

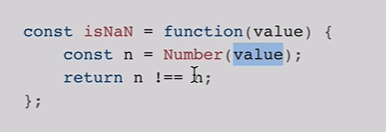


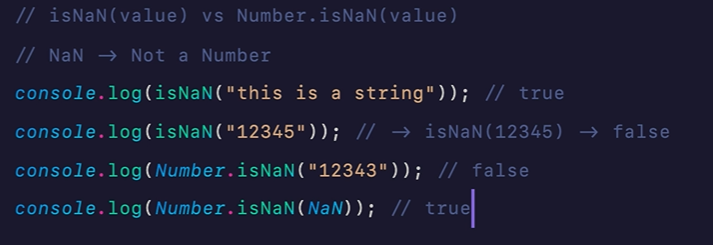
# Number

## Function



* parseInt(‘5’)
* parseFloat(‘5.2’)
* Math.abs(), math.floor(), math.ceil(), math.round(), Math.random() …
* toFixed(2) // Làm tròn tới 2 ký tự thập phân nhưng là string
* isNaN





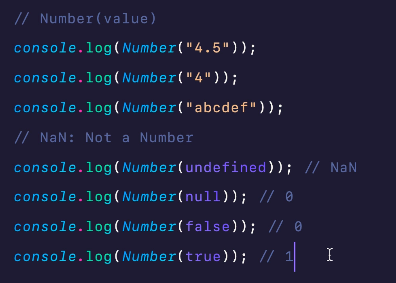
### Other data type

* undefined: VD khi khai báo 1 biến nhưng chưa gán giá trị, … (const phải gán giá trị ngay)
* null: nothing ,empty, …
* boolean: true and false
  + falsy : 0, “”, false, undefined, null, NaN
  + truthy

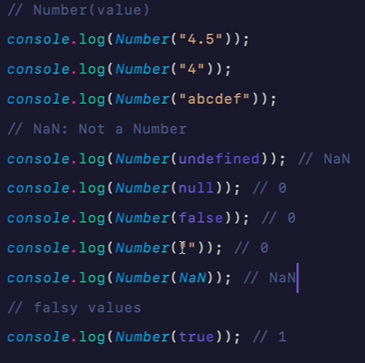
# Toán tử, câu điều kiện, …

## Hàm Number

* Number(value) : Nếu không phải số -> NaN ( Not a Number )

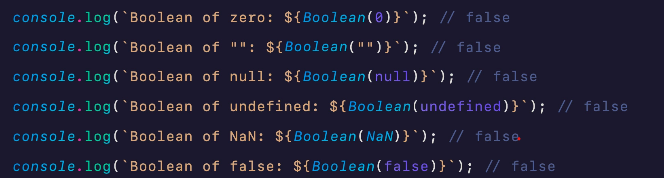


## Chuyển đổi dữ liệu về string ( Hàm string )



## Hàm boolean

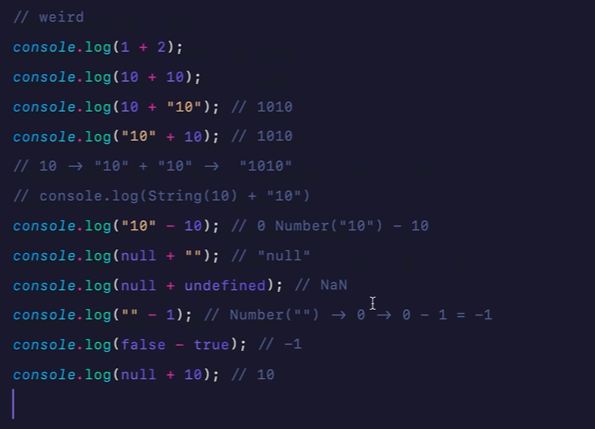
Các giá trị falsy trả về false



## Type coercion ( Ép kiểu )

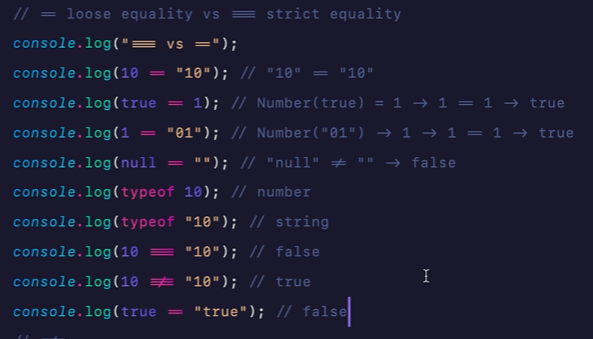
* Toán tử: + - \* /

Chỉ đặc biệt với dấu +

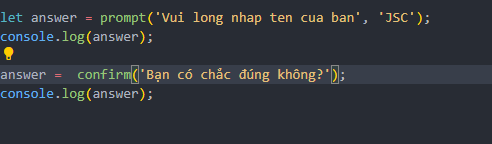


## Toán tử so sánh và toán tử logic

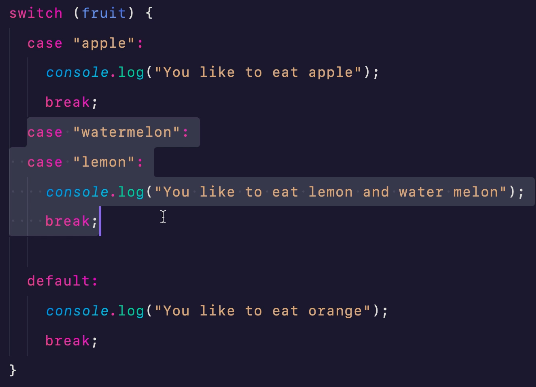
* > , < , >= , <=
* && , ||, !
* ==, !=, ===, !==, >==, <==



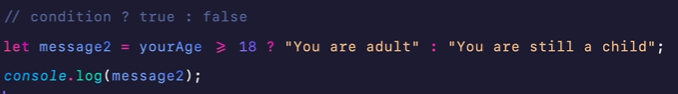
## Alert, prompt, confirm

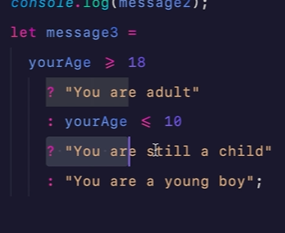


## Switch case



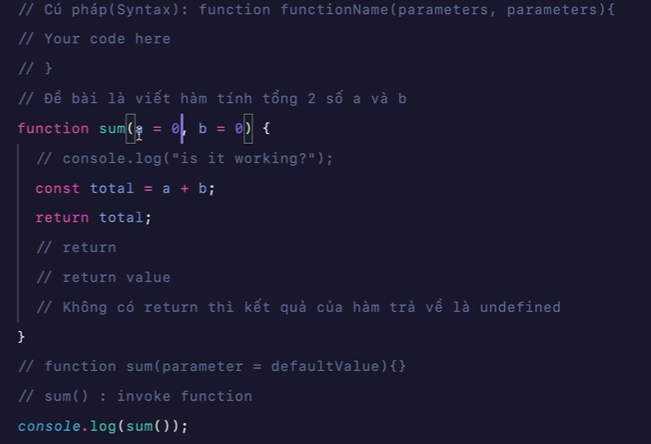
## Ternary operator

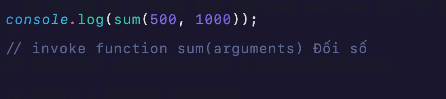




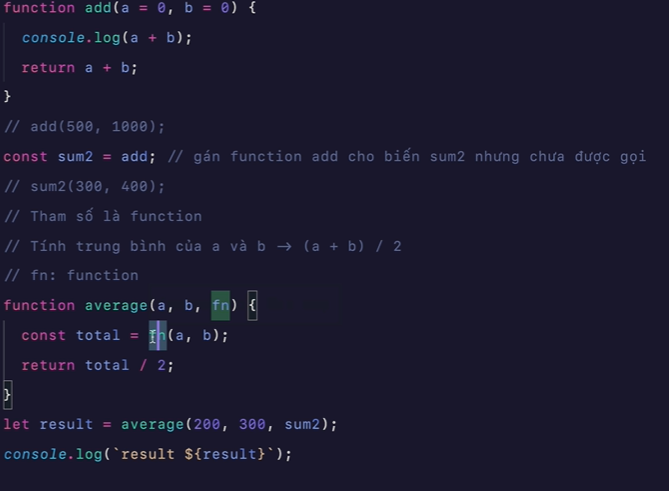
# Hàm toàn tập

## Hàm cơ bản



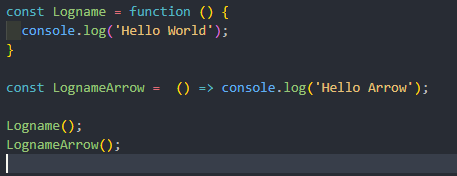


### Tham số là function

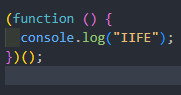


### Anonymous function (function expression)

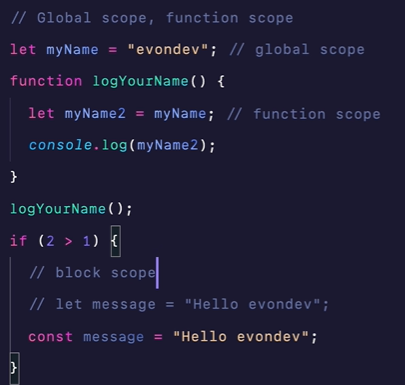
Không bị hoisting -> Phải khai báo trước.



### IIFE (Immediately invoked function execution)



### Global scope, function scope, block scope và từ khóa var



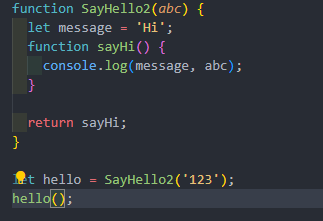


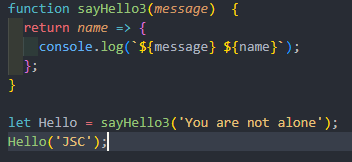
Var luôn là global scope và hoisting nên hạn chế sử dụng var

### Lexical Scope

### Closure

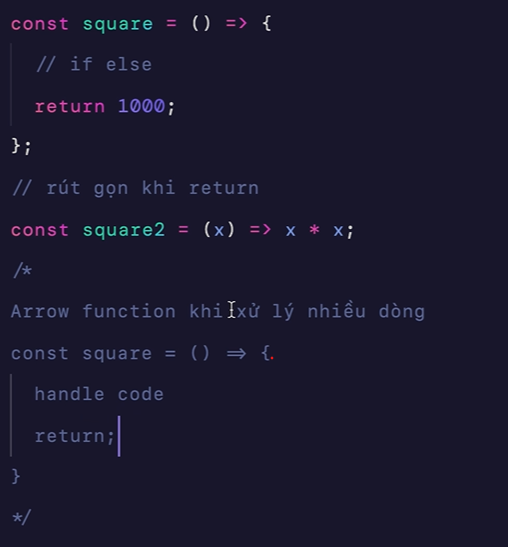
* Child function có thể truy xuất scope của parent function



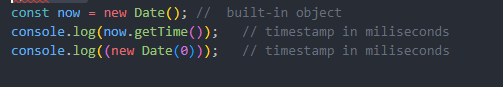


## Arrow function

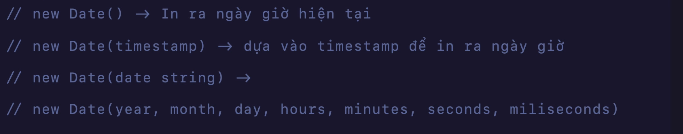
Là anonymous function(function )

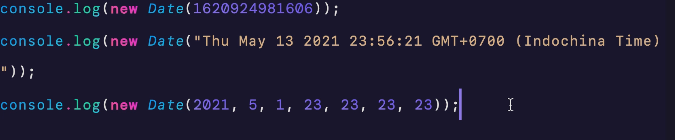


# Các hàm về thời gian

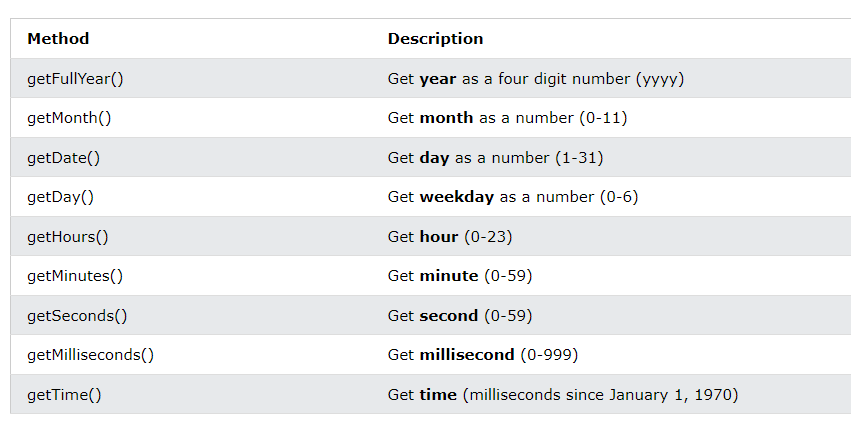


## 4 cách khởi tạo Date object cần biết

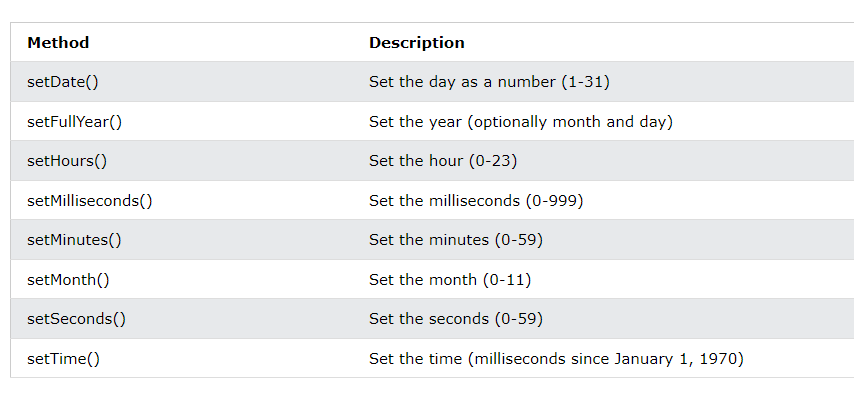




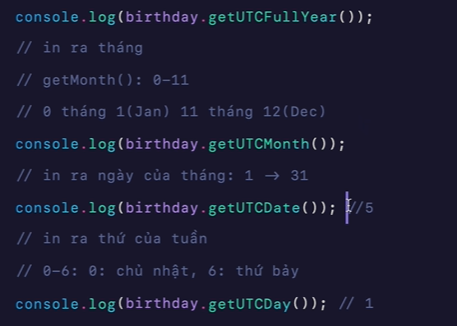
## Các hàm Get trong Date



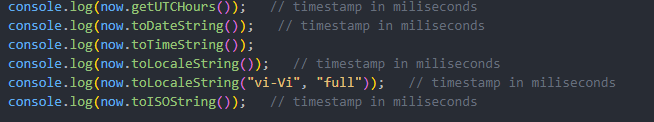
## Các hàm Set trong Date

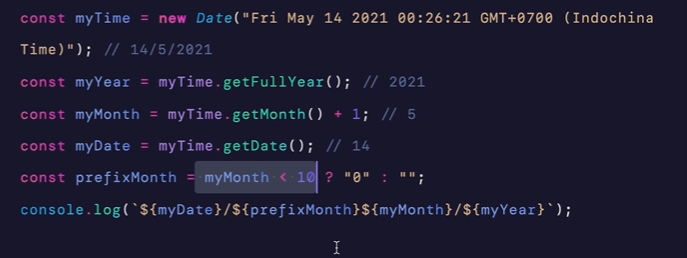


## UTC trong Date

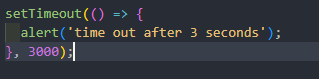


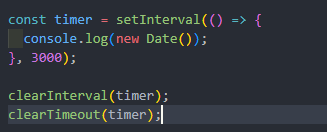
## Các hàm khác trong Date



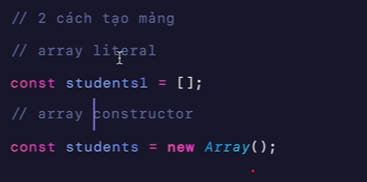


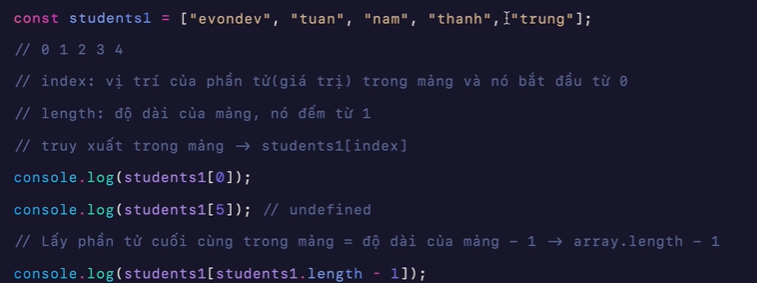
## Timer setTimeout và setInterval





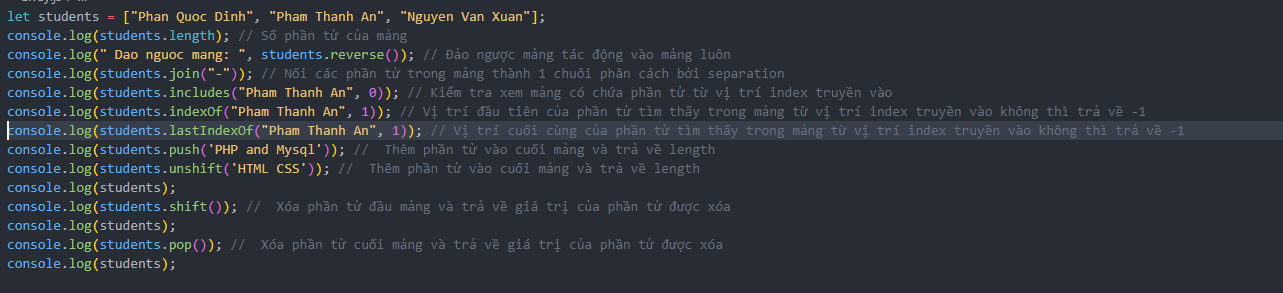
# Array





## Array method

* reverse(); // Đảo ngược mảng tác động vào mảng luôn
* join("-"); // Nối các phần tử trong mảng thành 1 chuôi phân cách bởi separation
* includes("Pham Thanh An", 0); // Kiểm tra xem mảng có chứa phần tử từ vị trí index truyền vào
* indexOf("Pham Thanh An", 1); // Vị trí đầu tiên của phần tử tìm thấy trong mảng từ vị trí index truyền vào không thì trả về -1
* lastIndexOf("Pham Thanh An", 1); // Vị trí cuối cùng của phần tử tìm thấy trong mảng từ vị trí index truyền vào không thì trả về -1
* push('PHP and Mysql'); //  Thêm phần tử vào cuối mảng và trả về length
* unshift('HTML CSS'); //  Thêm phần tử vào cuối mảng và trả về length
* shift(); //  Xóa phần tử đầu mảng và trả về giá trị của phần tử được xóa
* pop(); //  Xóa phần tử cuối mảng và trả về giá trị của phần tử được xóa

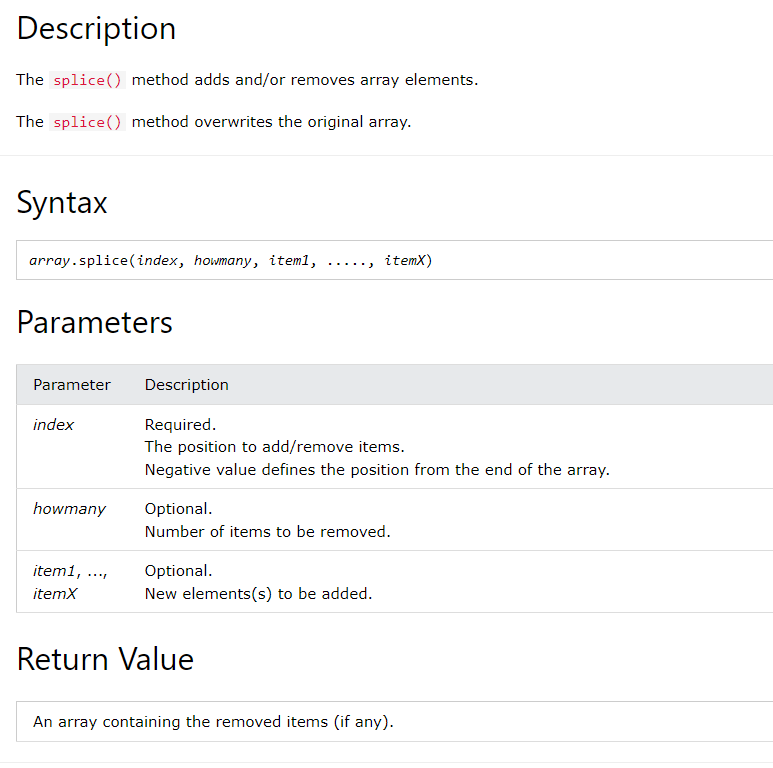


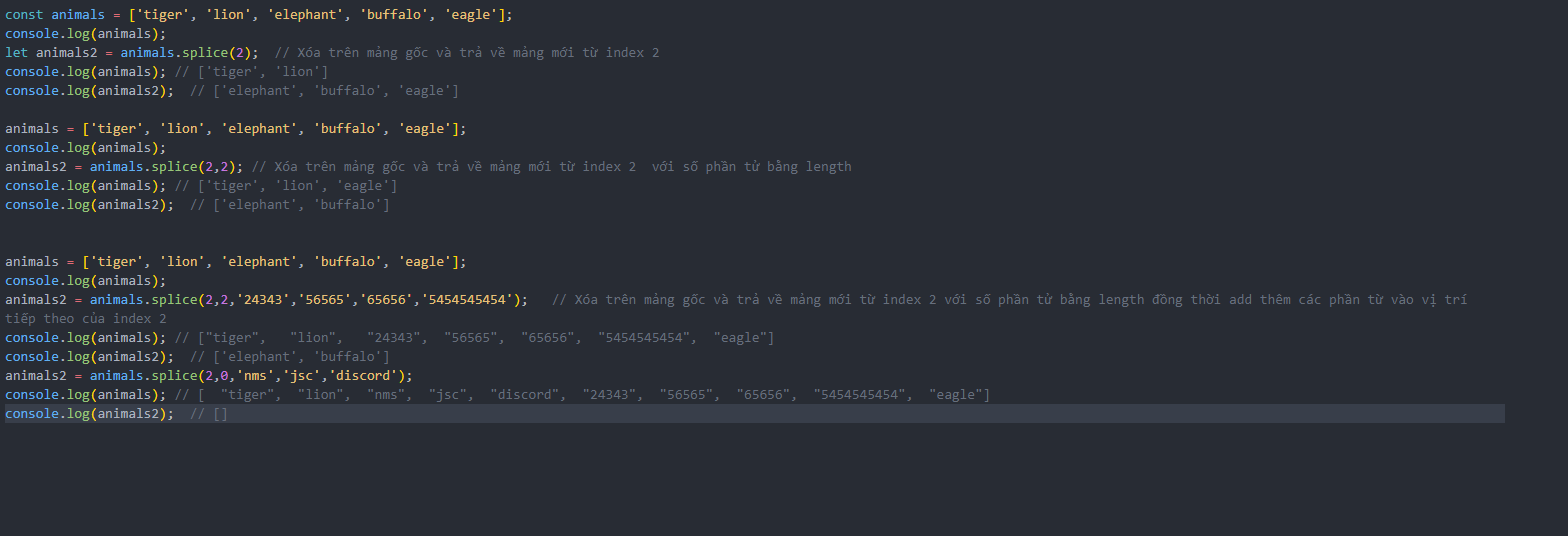
## slice method

Tạo ra mảng mới là copy của mảng ban đầu

* animals.slice() // Sao chép cả mảng
* animals.slice(2); // Sao chép từ index 2
* animals.slice(2,4); // Sao chep từ index start tới index end – 1
* const animals = ['tiger', 'lion', 'elephant', 'buffalo', 'eagle'];
* console.log(animals);
* let animals2 = animals.slice()   // Sao chép cả mảng
* console.log(animals2);
* animal2 = animals.slice(2);   // Sao chép từ index 2
* console.log(animal2);
* animal2 = animals.slice(2,4);  // Sao chep từ index start tới index end - 1
* console.log(animal2);

## splice method

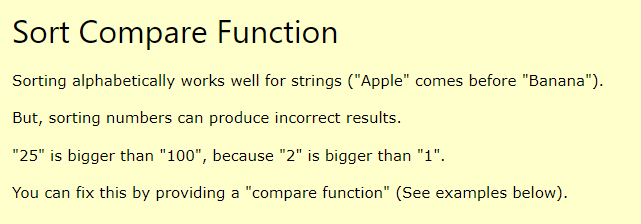


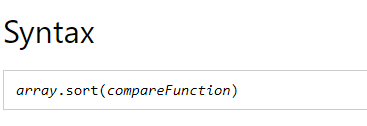


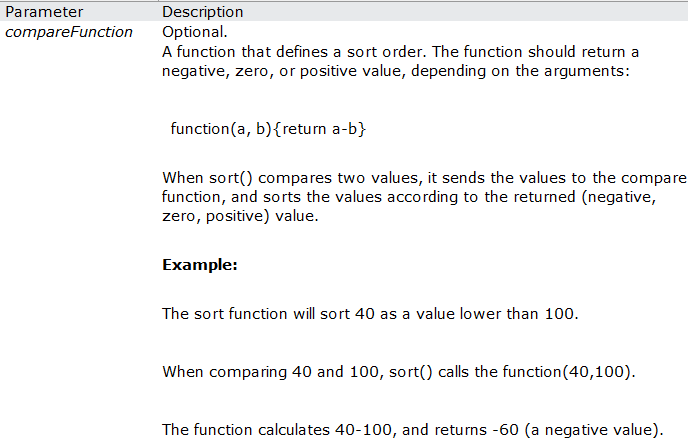
* splice(2); // Xóa trên mảng gốc và trả về mảng mới từ index 2
* splice(2,2); // Xóa trên mảng gốc và trả về mảng mới từ index 2 với số phần tử bằng length
* splice(2,2,'24343','56565','65656','5454545454'); // Xóa trên mảng gốc và trả về mảng mới từ index 2 với số phần tử bằng length đồng thời add thêm các phần từ vào vị trí tiếp theo của index 2
* splice(2,0,'nms','jsc','discord');

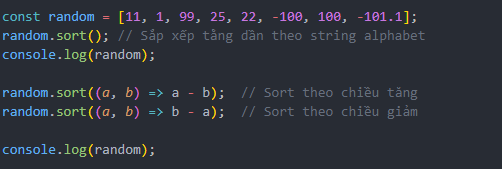
## sort method

<https://www.w3schools.com/jsref/jsref_sort.asp>



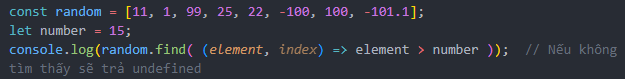






## find

Tìm phần tử đầu tiên trong mảng thỏa 1 điều kiện



## findIndex

Tương tự find nhưng trả về index (Nếu không tìm thấy sẽ trả về -1)

## map

Duyêt các phần tử trong mảng thực hiện 1 chức năng nào đó rồi trả lại mảng mới



## forEach

Cũng duyệt qua các phần tử trong mảng nhưng không có return. Thường dùng khi làm việc với DOM để xử lý chức năng nào đó trong mảng

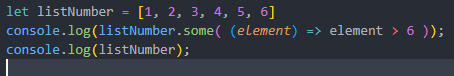
## Filter

Trả về hàm với các phần tử thỏa mãn 1 điều kiện nào đó



## Some và Every

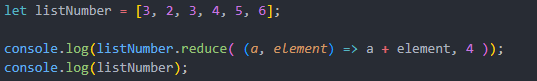
* Some check nếu có bất kỳ phần tử nào thỏa mãn (true)



* Every check nếu tất cả các phần tử thỏa mãn (true)



## Reduce



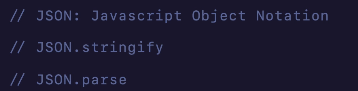
## Flat

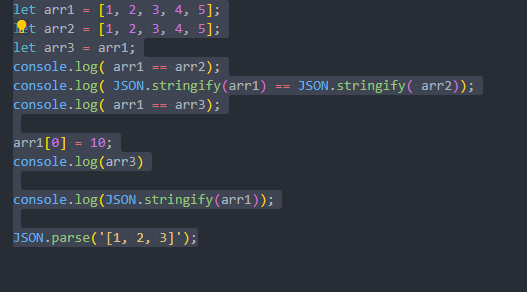
*/\*\*\*\*\* Flat array \*\*\*\*\*/  
console*.log('\*\*\*\*\* Flat array \*\*\*\*\*');  
const *arr* = [1, 3, [5,9], 8, 6, ['n', 'm', [9, 6, [1, 'nm']]]];  
*console*.log(*arr*);  
*console*.log(*arr*.flat(1));  
*console*.log(*arr*.flat(2));  
*console*.log(*arr*.flat(99));

## By value và by reference

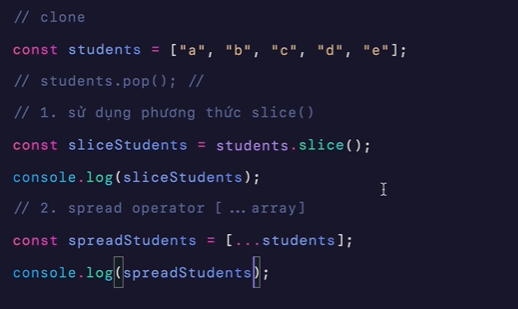
Cái dữ liệu nguyên thủy là by value còn lại là reference

### Compare array

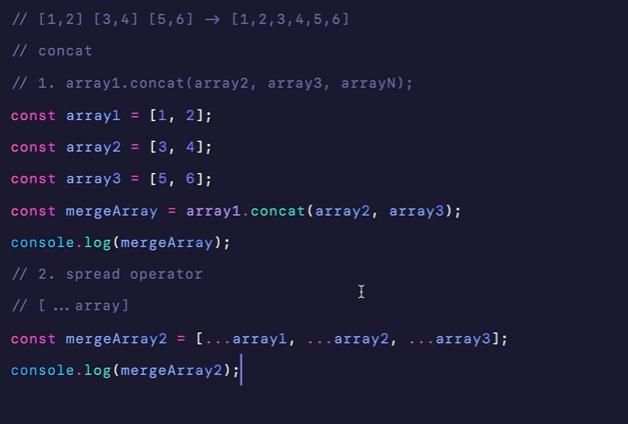




## 2 cách để sao chép mảng cần biết



## 2 cách để gộp mảng cần biết



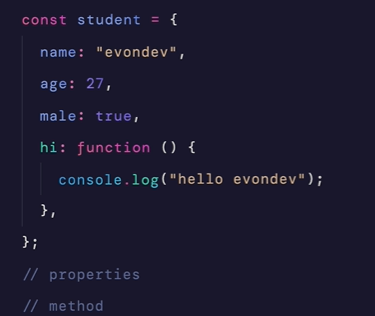
## Destructuring

Rest parameter

const *numbers* = [1, 2, 3, 4, 5, 6];  
  
for (let i = 0; i < *numbers*.length; i++ ) {  
 *console*.log(`Index ${i}: ${*numbers*[i]}`);  
}

# Object

Là 1 tập hợp các key: value



## Method Object.keys, Object.values, Object.entries

// object constructor  
const *objectConstructor* = new Object();  
  
const *nss* = {  
 1: 2,  
 3: 4,  
 ns: "nmms",  
 ss: {  
 ssx: 'ssx',  
 yyy: 'yyy'  
 }  
}  
  
// object.keys() trả về 1 mảng chứa các key của object  
*console*.log(*Object*.keys(*nss*)); // Trả về 1 array chứa key  
*console*.log(*Object*.values(*nss*)); // Trả về 1 array chứa values  
*console*.log(*Object*.entries(*nss*)); // Trả về 1 nested array mà mỗi phần tử là 1 array chứa key, values

## Object.assign

Gộp nhiều object vào với nhau

const *nss* = {  
 1: 2,  
 3: 4,  
 ns: "nmms",  
}  
  
const *nss2* = {  
 ss: 'ss',  
 yy: 'yy'  
}  
  
const *nss3* = {  
 ssy: 'ssy',  
 yyz: 'yyz'  
}  
  
  
*console*.log(*Object*.assign(*nss*, *nss3*, *nss2*));  
*console*.log({...*nss3*, ...*nss2*, ...*nss*});

## Object.freeze

Đóng băng không cho chỉnh sửa, thêm, bớt Object

const *nss* = {  
 1: 2,  
 3: 4,  
 ns: "nmms",  
}  
  
  
*nss*.nsx = '456';  
*nss*.nss = '123';  
*console*.log(*nss*);  
delete *nss*.nsx;  
*console*.log(*Object*.freeze(*nss*));  
  
*nss*.nss = '456';  
*console*.log(*nss*);  
  
delete *nss*.nss;  
*console*.log(*nss*);

## Object.seal

Cho phép chỉnh sửa key value nhưng không được thêm mới hoặc xóa

const *nss* = {  
 1: 2,  
 3: 4,  
 ns: "nmms",  
}  
  
  
*nss*.nsx = '456';  
*nss*.nss = '123';  
*console*.log(*nss*);  
delete *nss*.nsx;  
*nss*.nssxyz = '456';  
*console*.log(*Object*.seal(*nss*));  
  
*nss*.nss = '456';  
*console*.log(*nss*);  
  
delete *nss*.nss;  
*nss*.nssxy = '456';  
*console*.log(*nss*);

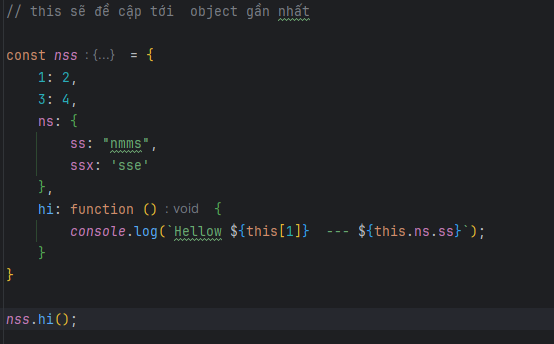
## Cách sao chép 1 object

Spread operator và Object.assign() (Chỉ copy được 1 cấp, từ cấp 2 trở đi vẫn bị by reference )

let *ss* = { ... *nss* };

## Từ khóa this trong Object

Trường hợp đơn giản nhất



## Optional chaining (?.)

const *nss* = {  
 1: 2,  
 3: 4,  
 ns: {  
 ss: "nmms",  
 ssx: 'sse'  
 },  
 hi: function () {  
 *console*.log(`Hellow ${this[1]} --- ${this.ns.ss}`);  
 },  
  
 // fullName: {  
 // name: 'you are'  
 // }  
}  
  
*console*.log(*nss*.fullName?.name);

## Destructoring

const *students* = {  
 name: 'Dinh',  
 surname: 'Phan',  
 hi: () => *console*.log(`Hello ${this.surname} ${this.name} `),  
 age: '39'  
}  
  
const {*name*, *age*, ...*nms*} = *students*;  
*console*.log(*name*, *age*, *nms*);  
  
function findInfo({ name,age,surname }) {  
 *console*.log(surname, name, age);  
}  
  
findInfo(*students*);

# DOM

DOCUMENT OBJECT MODEL

DOM node: tag (html, head, body, h1, …)

DOM attribute: href, src …

## Selecting Node

### document.querySelector

const *h1Content* = *document*.querySelector('#heading');

### document.querySelectorAll

// Trả về 1 Nodelist chứa danh sách các node. Nếu không có trả về empty  
// Có thẻ loop nhưng không phải array nên không dùng được các array method như push, shift, pop, ...  
const *all* = *document*.querySelectorAll('h2');  
*console*.log(*all*);

### document.getElementsByClassName

// Trả về 1 HTML Collection. Nếu không có trả về empty  
// // Có thẻ loop nhưng không phải array nên không dùng được các array method như push, shift, pop, ...  
const *list*= *document*.getElementsByClassName('list');  
*console*.log(*list*);

### document.getElementsByTagName

### document.getElementById

### getAttribute

// Chỉ dùng với 1 node  
const *attrNode* = *document*.querySelector('a');  
*console*.log( *attrNode*.getAttribute('href'));

*attrNode*.removeAttribute('class');  
*attrNode*.setAttribute('class2', 'nms');  
  
*console*.log(*attrNode*.hasAttribute('href'));  
*console*.log(*attrNode*.hasAttributes());

### textContent, innerText, innerHTML

// 1. textContent : Nội dung giữa các thẻ tag. Nếu có html tag nó vẫn lấy nội dung và remove thẻ tag đi và lấy cả khoảng trắng, xuống hàng  
const *spin* = *document*.querySelector('li:last-child');  
*console*.log(*spin*.textContent);  
  
// 2 .innerText: Loại bỏ hết khoảng trắng và xuống hàng 2 bên và cả html tag bên trong  
*console*.log(*spin*.innerText);  
  
//3. innerHTML: lấy toàn bộ nội dung kể cả html tag bên trong  
*console*.log(*spin*.innerHTML);

## Các phương thức về class

### classList.add, classList.remove, classList.contains, classList.toggle, select.className

const *listC* = *document*.querySelectorAll('.list:nth-child(2n)');  
*console*.log(*listC*);  
// selector.classList.add('is-green');  
for (let a of *listC*) {  
 a.classList.add('is-green');  
}  
  
// selector.classList.remove('is-green');  
for (let a of *listC*) {  
 a.classList.remove('list');  
}  
  
// selector.classList.contains('list');  
for (let a of *listC*) {  
 a.classList.contains('list');  
}  
  
const *listA* = *document*.querySelectorAll('.list');  
// selector.classList.toggle('list');  
for (let a of *listA*) {  
 a.classList.toggle('list');  
}

// select.className  
let *listB* = *document*.querySelectorAll('.list');  
for (let a of *listB*) {  
 *console*.log(a.className);  
}

for (let a of *listB*) {  
 a.className = '12323 54545454';  
}

### document.createElement và document.appendChild

const *h1* = *document*.createElement('h1');  
*document*.body.appendChild(*h1*);

const *body* = *document*.body;  
const *div* = *document*.createElement('div');  
*div*.classList.add("card");  
// document.body.appendChild(div);  
  
const *img* = *document*.createElement('img');  
// img.classList.add('card-image');  
*img*.setAttribute('class', 'card-image');  
*img*.setAttribute('src', 'https://source.unsplash.com/random')  
*img*.setAttribute('alt', 'random image');  
*div*.appendChild(*img*);  
*body*.appendChild(*div*);

### document.createTextNode

const *text* = *document*.createTextNode('Hello this is my H1 heading');  
const *h1* = *document*.createElement('h1');  
*h1*.appendChild(*text*);  
*body*.appendChild(*h1*);

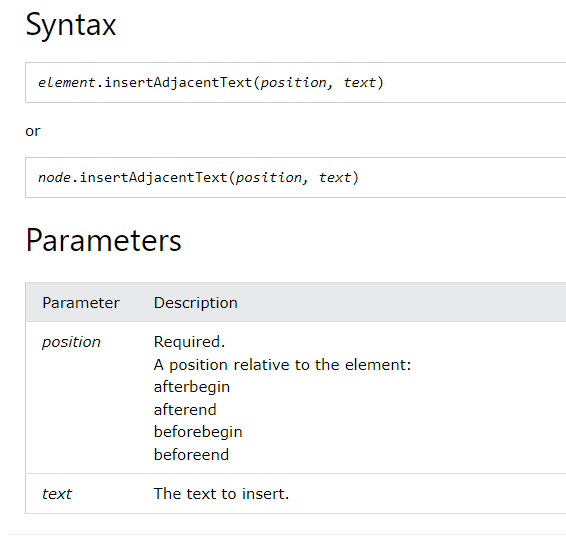
### cloneNode

const *h1Clone* = *h1*.cloneNode(true);  
*body*.appendChild(*h1Clone*);

### hasChildNodes

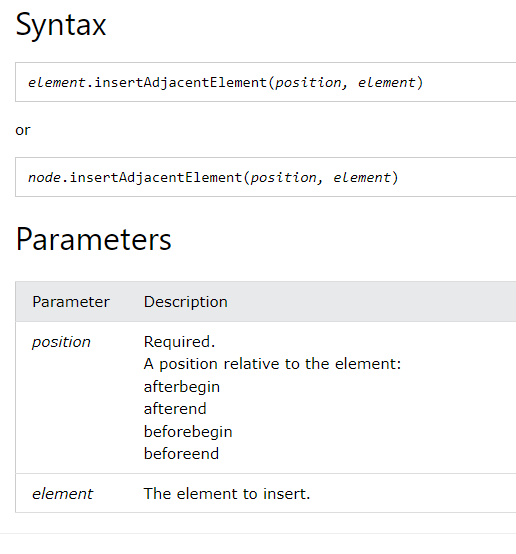
Kiểm tra xem element có phần tử con hay không? (Kể cả textNode)

### insertAdjacentText



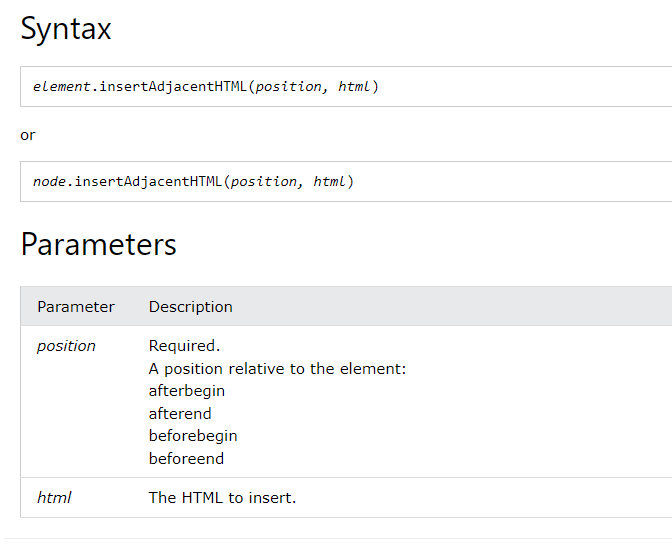
*h1*.insertAdjacentText('afterbegin', 'afte');  
*h1*.insertAdjacentText('beforebegin', '123');

### insertAdjacentElement



### insertAdjacentHTML

Chèn thêm 1 template string html



### parent Node, parent Element, removeChild

const *span* = *document*.body.querySelector('span');  
*console*.log(*span*.parentNode);  
*console*.log(*span*.parentElement);  
*console*.log(*span*.parentNode.parentNode);  
*console*.log(*span*.parentElement.parentElement);  
*span*.parentNode.removeChild(*span*);

### nextElementSibling, previousElementSibling

Trả về phần tử kế tiếp sau và trước

### childNodes, children, firstChild

childNodes: trả về các phần tử con bao gồm cả text node

children trả về node không bao gồm phần tử text node

firstChild: trả về phần tử con đầu tiên bao gồm text Node

firstElementChild: trả về phần tử elment con đầu tiên

lastChild

lastElementChild

### Offset

function print(value) {  
 *console*.log(value);  
}  
  
const *box* = *document*.querySelector('.box');  
  
print(*box*.offsetWidth); // Độ rộng của phần tử  
print(*box*.offsetHeight); // Chiều cao của phần tử  
print(*box*.offsetLeft); // Vị trí của nó so với bên trái  
print(*box*.offsetTop); // Vị trí của nó so với phía trên  
print(*box*.offsetParent); // Lấy ra phần tử cha để láy giá trị của phần tử cha

### Client

print(*box*.clientWidth); // Độ rộng của phần tử trừ border  
print(*box*.clientHeight); // Chiều cao của phần tử  
print(*box*.clientLeft); // Vị trí của nó so với bên trái tính cả border  
print(*box*.clientTop); // Vị trí của nó so với phía trên tính cả border

### Window

*window*.innerHeight  
*window*.outerHeight  
*window*.innerWidth  
*window*.outerWidth

## getBoundingClientRect()

# Sự kiện trong javascript

## Click

// addEventListener  
let *submit* = *document*.querySelector('.submit');  
*submit*.addEventListener('click', (e) => {  
 *submit*.classList.toggle('click');  
} )  
  
*submit*.addEventListener('click', click);  
  
function click(e) {  
 *submit*.classList.toggle('click');  
}

### Bubbling trong click

// addEventListener  
let *submit* = *document*.querySelector('.submit');  
const *span* = *document*.querySelector('.submit span')  
  
*span*.addEventListener('click', () => alert('span'));  
// submit.addEventListener('click', (e) => {  
// submit.classList.toggle('click');  
// } )  
  
*submit*.addEventListener('click', click);  
  
function click(e) {  
 *submit*.classList.toggle('click');  
}  
  
// bubbling Chạy click span (child) trước rồi chạy tiếp click submit (parent)

Sử dụng stopPropagation() để ngăn bubbling lên phần tử cha

// addEventListener  
let *submit* = *document*.querySelector('.submit');  
const *span* = *document*.querySelector('.submit span')  
  
*span*.addEventListener('click', (e) => {  
 alert('span');  
 e.stopPropagation();  
});  
// submit.addEventListener('click', (e) => {  
// submit.classList.toggle('click');  
// } )  
  
*submit*.addEventListener('click', click);  
  
function click(e) {  
 *submit*.classList.toggle('click');  
 e.stopPropagation();  
}  
  
*document*.body.addEventListener('click', () => *console*.log('body')  
);

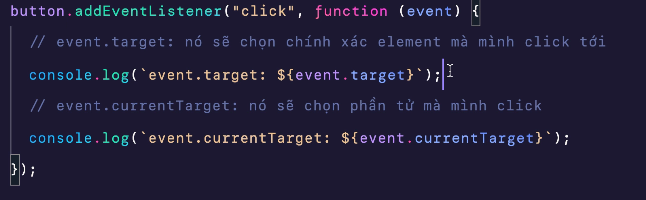
### Capturing trong click

*span*.addEventListener('click', (e) => {  
 alert('span 2');  
 // e.stopPropagation();  
 e.stopImmediatePropagation(); // The stopImmediatePropagation() method prevents other listeners of the same event from being called.  
});  
*span*.addEventListener('click', (e) => {  
 alert('span');  
 e.stopPropagation(); // method prevents propagation of the same event from being called.  
  
 // e.stopImmediatePropagation();  
});

Capture sẽ chạy ngược lại từ phần tử cha trước -> con

// addEventListener  
let *submit* = *document*.querySelector('.submit');  
const *span* = *document*.querySelector('.submit span')  
  
  
*span*.addEventListener('click', (e) => {  
 alert('span');  
  
}, { capture: true });  
  
  
*submit*.addEventListener('click', click, { capture: true });  
  
function click(e) {  
 *submit*.classList.toggle('click');  
}  
  
*document*.body.addEventListener('click', () => *console*.log('body'), { capture: true }  
);

### Target and currentTarget



### preventDefault()

Dùng để ngăn chặn 1 hành vi mặc định nào đó

