**JAVA SCRIPT**

Mục lục

[**1.** **JS cơ bản** 5](#_Toc85190126)

[**1.1** **JS có thể thay đổi được nội dung trong HTML** 5](#_Toc85190127)

[**1.2** **Nơi sử dụng JS** 5](#_Toc85190128)

[**1.3** **JS Output** 5](#_Toc85190129)

[**1.4** **Lệnh trong JS** 6](#_Toc85190130)

[**1.5** **Cú pháp (syntax) JS** 6](#_Toc85190131)

[**1.6** **Biến trong JS** 6](#_Toc85190132)

[**1.7 Let** 7](#_Toc85190133)

[**1.8 Const** 7](#_Toc85190134)

[**1.9 Kiểu dữ liệu JS** 8](#_Toc85190135)

[**1.10 Cú pháp Function** 8](#_Toc85190136)

[**1.11 Cú pháp Object** 8](#_Toc85190137)

[**1.12 Sự kiên JS** 8](#_Toc85190138)

[**1.13 JS String** 8](#_Toc85190139)

[**1.14 Phương thức String** 9](#_Toc85190140)

[**Tách các bộ phận trong chuỗi** 9](#_Toc85190141)

[**Convert Upper và Lower Case** 10](#_Toc85190142)

[**Phương thức concat()** 10](#_Toc85190143)

[**Phương thức trim()** 10](#_Toc85190144)

[**Phương thức String Padding** 10](#_Toc85190145)

[**Phương thức charAt()** 10](#_Toc85190146)

[**Phương thức charCodeAt()** 11](#_Toc85190147)

[**Truy cập Property** 11](#_Toc85190148)

[**Convert String sang Array** 11](#_Toc85190149)

[**1.15 Phương thức Searching** 11](#_Toc85190150)

[**1.16 Chữ mẫu String (Template Literals)** 13](#_Toc85190151)

[**1.17 Kiểu số Number JS** 13](#_Toc85190152)

[NaN – Not a number 14](#_Toc85190153)

[**1.18 Các phương thức về number** 15](#_Toc85190154)

[Convert number sang string 15](#_Toc85190155)

[Phương thức toExponential() 16](#_Toc85190156)

[Phương thức toFixed() 16](#_Toc85190157)

[Phương thức toPrecision() 16](#_Toc85190158)

[Phương thức Number() 16](#_Toc85190159)

[**1.19 Phương thức Mảng JS** 16](#_Toc85190160)

[**Thuộc tính mảng** 17](#_Toc85190161)

[**Convert mảng qua string - .toString()** 17](#_Toc85190162)

[**Phương thức nối chuỗi qua ký tự** 17](#_Toc85190163)

[**Phương thức pop() – loại ra một phần tử nằm cuối** 17](#_Toc85190164)

[**Phương thức push() – thêm một phần tử nằm cuối** 18](#_Toc85190165)

[**Phương thức shift() – loại bỏ một phần tử nằm đầu** 18](#_Toc85190166)

[**Phương thức unsift – thêm một phần tử vào đầu** 18](#_Toc85190167)

[**Xoá phần tử trong mảng** 18](#_Toc85190168)

[**Phương thức splice()** 18](#_Toc85190169)

[**Phương thức concat()** 18](#_Toc85190170)

[**Phương thức slice()** 19](#_Toc85190171)

[**Phương thức toString()** 19](#_Toc85190172)

[**Phương thức copyWithin()** 19](#_Toc85190173)

[**Phương thức entries()** 19](#_Toc85190174)

[**Phương thức every()** 19](#_Toc85190175)

[**Phương thức fill()** 20](#_Toc85190176)

[**Phương thức filter()** 20](#_Toc85190177)

[**Phương thức find()** 20](#_Toc85190178)

[**Phương thức findIndex()** 21](#_Toc85190179)

[**Phương thức foreach()** 21](#_Toc85190180)

[**Phương thức from()** 21](#_Toc85190181)

[**Phương thức includes()** 22](#_Toc85190182)

[**Phương thức indexOf()** 22](#_Toc85190183)

[**Phương thức isArray()** 22](#_Toc85190184)

[**Phương thức keys()** 22](#_Toc85190185)

[**Phương thức lastIndexOf()** 22](#_Toc85190186)

[**Phương thức map()** 22](#_Toc85190187)

[**Phương thức reduce()** 23](#_Toc85190188)

[**Phương thức reduceRight()** 24](#_Toc85190189)

[**Phương thức reserve()** 24](#_Toc85190190)

[**Phương thức some()** 24](#_Toc85190191)

[**Phương thức sort()** 24](#_Toc85190192)

[**2.** **JS Object** 25](#_Toc85190193)

[**3.** **JS Functions** 25](#_Toc85190194)

[**4.** **JS Classes** 25](#_Toc85190195)

[**5.** **JS Async** 25](#_Toc85190196)

[**5.1 Trình tự các hàm** 25](#_Toc85190197)

[**5.2 Callback** 25](#_Toc85190198)

[**5.3 JS không đồng bộ** 25](#_Toc85190199)

[**setTimeout()** 25](#_Toc85190200)

[**setIntervals()** 25](#_Toc85190201)

[**Waiting for files** 26](#_Toc85190202)

[**5.4 JS Promises** 26](#_Toc85190203)

[**Promise Syntax** 26](#_Toc85190204)

[**Thuộc tính trong Promise** 27](#_Toc85190205)

[**Waiting for a Timeout, sử dụng Promise** 27](#_Toc85190206)

[**Waiting for file, sử dụng promise** 28](#_Toc85190207)

[**5.5 JS Async/Await** 28](#_Toc85190208)

[**Async** 29](#_Toc85190209)

[**Await Syntax** 29](#_Toc85190210)

[**Waiting for a File** 30](#_Toc85190211)

[**6.** **JS HTML DOM** 30](#_Toc85190212)

[**7.** **JS Brower BOM** 30](#_Toc85190213)

[**8.** **JS Web APIs** 30](#_Toc85190214)

[**9.** **JS Ajax** 30](#_Toc85190215)

[**10.** **JS Json** 30](#_Toc85190216)

1. **JS cơ bản**

## **JS có thể thay đổi được nội dung trong HTML**

In ra màn hình dòng chữ

document.getElementById('demo').innerHTML = 'Hello JavaScript';

Thay đổi Css

document.getElementById("demo").style.fontSize = "35px";

Show/Hide một elements sử dụng display style

document.getElementById("demo").style.display = "none";

document.getElementById("demo").style.display = "block";

## **Nơi sử dụng JS**

Có thể sử dụng trực tiếp vào thẻ như cách trên

Cách thứ 2 có thể định nghĩ qua hàm

<button type="button" onclick="myFunction()">Try it</button>

Và hàm này sẽ được định nghĩa có thể trên head hoặc ở dưới thông qua thẻ <script>

<script>  
function myFunction() {  
  document.getElementById("demo").innerHTML = "Paragraph changed.";  
}  
</script>

Hoặc định nghĩa ở một file khác và add đường dẫn vào, file này sẽ chứa hàm xử lý

<script src="myScript.js"></script>

* 1. **JS Output**

JS có thể in dữ liệu ra ngoài bằng các cách sau:

* In ra ngoài màn hình innerHTML và document.write()

document.getElementById("demo").innerHTML = 5 + 6;

document.write(5 + 6);

* In ra thông báo window.alert()

alert(5 + 6);

* In ra màn hình debug console.log()

console.log(5 + 6);

* Lệnh hiện xuất bản in trang html window.print()

<button onclick="window.print()">Print this page</button>

* 1. **Lệnh trong JS**

Câu lệnh JS có thể bao gồm:

* Giá trị
* Toán tử
* Biểu thức
* Từ khoá
* Nhận xét

**Từ khoá (keywords) trong JS:**

* var : Khai báo một biến toàn cục
* let : Khai báo một biến trong một khối block
* const : Khai báo một hằng số không đổi
* if – if else – else : rẽ nhánh điều kiện
* switch – case : đánh dấu các khối thực thi các câu lệnh khác nhau
* for: khối vòng lặp
* function: định nghĩa 1 hàm
* return: trả về giá trị cho 1 hàm
* try – catch: xử lý trường hợp lỗi trong 1 khối

Các từ dành riêng cho JS: <https://www.w3schools.com/js/js_reserved.asp>

## **Cú pháp (syntax) JS**

**Mã định danh (Identifiers) JS**

Ký tự đầu tiên là chữ cái, hoặc dấu \_ hoặc dấu $

## **Biến trong JS**

Mặc định kiểu biến là var và giá trị là undefined

var carName;

**Khai báo lại các biến JS (chỉ var)**

var carName = "Volvo";  
var carName;

* Khi in ra thì sẽ chọn các biến có giá trị khác undefined

**Phân biệt kết quả giữa các phép cộng**

Khi có một chuỗi được cộng vào thì kết quả cộng sau đó sẽ thành chuỗi, ngược lại nếu phát hiện được là số thì sẽ cộng như bth

VD:

var x = "John" + " " + "Doe"; => John Doe

var x = "5" + 2 + 3; => 523

var x = 2 + 3 + "5"; => 55

## **1.7 Let**

- Không thể sử dụng khai báo lại giống như kiểu var

- Các biến được được định nghĩa với let phải được khai báo trước

- Các biến được xác định trong phạm vi khối

## **1.8 Const**

- Không thể thay đổi giá trị kiểu const trong cùng khối

- Có thể định nghĩa lại biến const cùng tên ở khối khác

const x = 10;  
// Here x is 10  
  
{  
const x = 2;  
// Here x is 2  
}  
  
// Here x is 10

var x = 2;     // Allowed  
const x = 2;   // Not allowed  
  
{  
let x = 2;     // Allowed  
const x = 2;   // Not allowed  
}  
  
{  
const x = 2;   // Allowed  
const x = 2;   // Not allowed  
}

**Const Hoisting**

* Cơ chế hoisting ở const

alert (carName);  
const carName = "Volvo"; => ReferenceError

Khác với var sẽ không lỗi vì mặc định sẽ là var, còn khi const được khai báo sau mà in trước thì sẽ lỗi

## **1.9 Kiểu dữ liệu JS**

let length = 16;                               // Number  
let lastName = "Johnson";                      // String  
let x = {firstName:"John", lastName:"Doe"};    // Object

Thể loại động

let x;           // Now x is undefined  
x = 5;           // Now x is a Number  
x = "John";      // Now x is a String

## **1.10 Cú pháp Function**

function name(parameter1, parameter2, parameter3) {  
  // code to be executed  
}

## **1.11 Cú pháp Object**

Cách truy xuất đến các property:

* *objectName.propertyName*
* *objectName["propertyName"]*

Phương thức là hàm được định nghĩa trong object đc xem như 1 property

const person = {  
  firstName: "John",  
  lastName : "Doe",  
  id       : 5566,  
  fullName : function() {  
    return this.firstName + " " + this.lastName;  
  }  
};

**Từ khoá this**

* Truy xuất đến các property trong object

## **1.12 Sự kiên JS**

Cú pháp:

<element event=**"*some JavaScript*"**>

Tất cả các event: <https://www.w3schools.com/jsref/dom_obj_event.asp>

## **1.13 JS String**

- Dùng property length để đếm độ dài chuỗi

- Sử dụng các dấu gạch chéo tồn tại trong chuỗi

* let text = "We are the so-called \"Vikings\" from the north.";

// We are the so-called "Vikings" from the north.

* let text= 'It\'s alright.';

// It’s alright.

* let text = "The character \\ is called backslash.";

// The character \ is called backslash.

* 6 chuỗi xử lý hợp lệ trong JS:

|  |  |
| --- | --- |
| \b | Cách |
| \f | Form Feed |
| \n | Xuống dòng |
| \r | Carriage Return |
| \t | Tab ngang |
| \v | Tab dọc |

* Không nên dùng từ khoá new để tạo lớp String

Khi so sánh sử dụng dấu === thì sẽ cho ra kết quả luôn false khi so sánh 2 Object

## **1.14 Phương thức String**

### **Tách các bộ phận trong chuỗi**

* Slice(start, end)

let str = "Apple, Banana, Kiwi";  
str.slice(7, 13)     // Returns Banana

let str = "Apple, Banana, Kiwi";  
str.slice(-12, -6)    // Returns Banana

str.slice(7);    // Returns Banana,Kiwi

str.slice(-12)    // Returns Banana,Kiwi

* Substring(start, end) – giống với slice

let str = "Apple, Banana, Kiwi";  
substring(7, 13)    // Returns Banana

* Substr(start, length)

let str = "Apple, Banana, Kiwi";  
str.substr(7, 6)    // Returns Banana

let str = "Apple, Banana, Kiwi";  
str.substr(7)    // Returns Banana,Kiwi

let str = "Apple, Banana, Kiwi";  
str.substr(-4)    // Returns Kiwi

* Replace()

let text = "Please visit Microsoft and Microsoft!";  
let newText = text.replace("Microsoft", "W3Schools");//change one

let text = "Please visit Microsoft!";  
let newText = text.replace("MICROSOFT", "W3Schools");// no change

let text = "Please visit Microsoft!";  
let newText = text.replace(/MICROSOFT/i, "W3Schools");//change

let text = "Please visit Microsoft and Microsoft!";  
let newText = text.replace(/Microsoft/g, "W3Schools");// change all

### **Convert Upper và Lower Case**

let text1 = "Hello World!";       // String  
let text2 = text1.toUpperCase();  // text2 is text1 converted to upper

let text1 = "Hello World!";       // String  
let text2 = text1.toLowerCase();  // text2 is text1 converted to lower

### **Phương thức concat()**

text = "Hello" + " " + "World!";  
text = "Hello".concat(" ", "World!");// Hello World!

### **Phương thức trim()**

let text = "       Hello World!        ";  
text.trim()    // Returns "Hello World!"

### **Phương thức String Padding**

let text = "5";  
text.padStart(4,0)    // Returns 0005

let text = "5";  
text.padEnd(4,0)     // Returns 5000

### **Phương thức charAt()**

let text = "HELLO WORLD";  
text.charAt(0)           // Returns H

### **Phương thức charCodeAt()**

* The method returns a UTF-16 code (an integer between 0 and 65535).

let text = "HELLO WORLD";  
  
text.charCodeAt(0)       // Returns 72

### **Truy cập Property**

let text = "HELLO WORLD";  
text[0]                   // returns H

let text = "HELLO WORLD";  
text[0] = "A"             // Gives no error, but does not work  
text[0]                   // returns H

### **Convert String sang Array**

* Sử dụng phương thức split() để cắt chuỗi ra thành mảng

text.split(",")          // Split on commas  
text.split(" ")          // Split on spaces  
text.split("|")          // Split on pipe

VD:

<script>

let text = "Hello";

const myArr = text.split("");

text = "";

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

text += myArr[i] + "<br>"

}

document.getElementById("demo").innerHTML = text;

</script>

* H
* e
* l
* l
* o

## **1.15 Phương thức Searching**

* indexOf()

let str = "Please locate where 'locate' occurs!";  
str.indexOf("locate")    // Returns 7

* lastIndexOf()

let str = "Please locate where 'locate' occurs!";  
str.lastIndexOf("locate")    // Returns 21 when Not Found return -1

let str = "Please locate where 'locate' occurs!";  
str.indexOf("locate", 15)    // Returns 21 (chỉ số đầu vào nhỏ hơn hoặc bằng chỉ số đúng)

* search()

let str = "Please locate where 'locate' occurs!";  
str.search("locate")     // Returns 7

* match()

let text = "The rain in SPAIN stays mainly in the plain";  
text.match(/ain/g)    // Returns an array [ain,ain,ain]

let text = "The rain in SPAIN stays mainly in the plain";  
text.match(/ain/gi)   // Returns an array [ain,AIN,ain,ain]

* includes()

let text = "Hello world, welcome to the universe.";  
text.includes("world")    // Returns true

let text = "Hello world, welcome to the universe.";  
text.includes("world", 12)    // Returns false

* startsWith()

let text = "Hello world, welcome to the universe.";  
  
text.startsWith("Hello")   // Returns true

let text = "Hello world, welcome to the universe.";  
  
text.startsWith("world")    // Returns false

let text = "Hello world, welcome to the universe.";  
  
text.startsWith("world", 5)    // Returns false

* endsWith()

var text = "John Doe";  
text.endsWith("Doe")    // Returns true

## **1.16 Chữ mẫu String (Template Literals)**

**Cú pháp Back-Tics**

- Chữ mẫu sử dụng dấu ``

let text = `Hello World!`;

* Biến thay thế

let firstName = "John";  
let lastName = "Doe";  
  
let text = `Welcome ${firstName}, ${lastName}!`; //Welcome John, Doe!

* Thay thế biểu thức

let price = 10;  
let VAT = 0.25;  
  
let total = `Total: ${(price \* (1 + VAT)).toFixed(2)}`; //Total: 12.50

* HTML Templates

let header = "Templates Literals";  
let tags = ["template literals", "javascript", "es6"];  
  
let html = `<h2>${header}</h2><ul>`;  
for (const x of tags) {  
  html += `<li>${x}</li>`;  
}  
  
html += `</ul>`;

Text

Description automatically generated

## **1.17 Kiểu số Number JS**

Số lớn hay số nhỏ thường được ký hiệu khoa học (e)

let x = 123e5;    // 12300000  
let y = 123e-5;   // 0.00123

**Độ chính xác**

let x = 999999999999999;   // x will be 999999999999999  
let y = 9999999999999999;  // y will be 10000000000000000

Số thập phân tối đa là 17, nhưng số học dấu phẩy động không phải lúc nào cũng chính xác 100%

let x = 0.2 + 0.1;         // x will be 0.30000000000000004

Để giải quyết vấn đề này, dùng phép nhân và phép cộng

let x = (0.2 \* 10 + 0.1 \* 10) / 10;       // x will be 0.3

Phát hiện phép toán (trừ phép cộng) sẽ tự động đổi kiểu dữ liệu sang số để tính toán (đối với tất cả thành phần là chuỗi dạng số)

let x = "100";  
let y = "10";  
let z = x / y;       // z will be 10

let x = "100";  
let y = "10";  
let z = x \* y;       // z will be 1000

let x = "100";  
let y = "10";  
let z = x - y;       // z will be 90

### NaN – Not a number

let x = 100 / "Apple";  // x will be NaN (Not a Number)

let x = 100 / "10";     // x will be 10

Sử dụng hàm isNaN() để kiểm tra có phải số ko?

let x = 100 / "Apple";  
isNaN(x);               // returns true because x is Not a Number

let x = NaN;  
let y = 5;  
let z = x + y;         // z will be NaN

let x = NaN;  
let y = "5";  
let z = x + y;         // z will be NaN5

<script>

let x = NaN;

document.getElementById("demo").innerHTML = typeof x; // number

</script>

**Infinity (hoặc -Infinity)**

* Sử dụng Infinity (hoặc -Infinity) trong trường hợp trả về số lớn nhất hoặc số nhỏ nhất

let myNumber = 2;  
while (myNumber != Infinity) {   // Execute until Infinity  
  myNumber = myNumber \* myNumber;  
}

4  
16  
256  
65536  
4294967296  
18446744073709552000  
3.402823669209385e+38  
1.157920892373162e+77  
1.3407807929942597e+154  
Infinity

**Phép chia số 0 cũng bằng Infinity**

let x =  2 / 0;       // x will be Infinity  
let y = -2 / 0;       // y will be -Infinity

TypeOf của Infitity là number

**Số Hex**

let x = 0xFF;        // x will be 255

Đổi hệ cơ số:

let myNumber = 32;  
myNumber.toString(10);  // returns 32  
myNumber.toString(32);  // returns 10  
myNumber.toString(16);  // returns 20  
myNumber.toString(8);   // returns 40  
myNumber.toString(2);   // returns 100000

32 =  
Decimal 32  
Hexadecimal 20  
Octal 40  
Binary 100000

## **1.18 Các phương thức về number**

### Convert number sang string

let x = 123;  
x.toString();            // returns 123 from variable x  
(123).toString();        // returns 123 from literal 123  
(100 + 23).toString();   // returns 123 from expression 100 + 23

### Phương thức toExponential()

* Làm tròn số sau dấu thập phân, trả vể kiểu string

let x = 9.656;  
x.toExponential(2);     // returns 9.66e+0  
x.toExponential(4);     // returns 9.6560e+0  
x.toExponential(6);     // returns 9.656000e+0

### Phương thức toFixed()

* Làm tròn số sau dấu thập phân, trả về kiểu string

let x = 9.656;  
x.toFixed(0);           // returns 10  
x.toFixed(2);           // returns 9.66  
x.toFixed(4);           // returns 9.6560  
x.toFixed(6);           // returns 9.656000

### Phương thức toPrecision()

* Làm tròn số , theo độ dài được chỉ định, trả về kiểu string

let x = 9.656;  
x.toPrecision();        // returns 9.656  
x.toPrecision(2);       // returns 9.7  
x.toPrecision(4);       // returns 9.656  
x.toPrecision(6);       // returns 9.65600

### Phương thức Number()

Number(true);          // returns 1  
Number(false);         // returns 0  
Number("10");          // returns 10  
Number("  10");        // returns 10  
Number("10  ");        // returns 10  
Number(" 10  ");       // returns 10  
Number("10.33");       // returns 10.33  
Number("10,33");       // returns NaN  
Number("10 33");       // returns NaN  
Number("John");        // returns NaN

* Number() cũng có thẻ convert từ kiểu Date sang, tính theo đơn vị milliseconds kểu từ 01-1-1970

// This returns 0://start  
Number(new Date("1970-01-01"))

// This returns 86400000//   
Number(new Date("1970-01-02"))

## **1.19 Phương thức Mảng JS**

### **Thuộc tính mảng**

* Constructor: trả về hàm đã tạo của đối tượng Array

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.constructor;  
// Returns function Array() { [native code] }

* Length
* Prototype: cho phép thêm thuộc tính và phương thức vào đối tượng mảng

<script>

// Add a new method

Array.prototype.myUcase = function() {

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

this[i] = this[i].toUpperCase();

}

};

// Use the method on any array

const fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.myUcase();

document.getElementById("demo").innerHTML = fruits;// BANANA,ORANGE,APPLE,MANGO

</script>

### **Convert mảng qua string - .toString()**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.toString();

Banana,Orange,Apple,Mango

### **Phương thức nối chuỗi qua ký tự**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.join(" \* ");

Banana \* Orange \* Apple \* Mango

### **Phương thức pop() – loại ra một phần tử nằm cuối**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
let x = fruits.pop();  // x = "Mango"

### **Phương thức push() – thêm một phần tử nằm cuối**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
let x = fruits.push("Kiwi");   //  x = 5

### **Phương thức shift() – loại bỏ một phần tử nằm đầu**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
let x = fruits.shift();    // x = "Banana"

### **Phương thức unsift – thêm một phần tử vào đầu**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.unshift("Lemon");    // Returns 5

### **Xoá phần tử trong mảng**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
delete fruits[0];           // Changes the first element in fruits to **undefined**

### **Phương thức splice()**

const fruits = ["Banana", "Orange", "Apple", "Mango"];

* *Kiểu 1:*

fruits.splice(2, 0, “Lemon”, “Kiwi”);

2: Thêm các giá trị mới vào trước chỉ số [2]

0: Là chỉ mục cờ bắt đầu thêm giá trị mảng vào sau ở lần thứ 2->n(lần thứ nhất sẽ thêm hết)

“Lemon”: Các giá trị sẽ được định thêm vào

* *Kiểu 2:*

fruits.splice(từ đâu, đến đâu);   // Removes the first element

### **Phương thức concat()**

const myGirls = ["Cecilie", "Lone"];  
const myBoys = ["Emil", "Tobias", "Linus"];  
  
// Concatenate (join) myGirls and myBoys  
const myChildren = myGirls.concat(myBoys);

const arr1 = ["Cecilie", "Lone"];  
const arr2 = ["Emil", "Tobias", "Linus"];  
const arr3 = ["Robin", "Morgan"];  
const myChildren = arr1.concat(arr2, arr3);

### **Phương thức slice()**

const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  
const citrus = fruits.slice(1);// cắt bỏ phần tử 1 ra

### **Phương thức toString()**

* JS tự động convert mảng qua kiểu string cách nhau bởi dấu ,

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.toString();// hoặc fruits

### **Phương thức copyWithin()**

<script>

const fruits = ["Banana", "Orange", "Apple", "Mango"];

document.getElementById("demo").innerHTML = fruits.copyWithin(2,1);//số đầu là vị trí chỉ mục thay thê, số hai là giá trị sẽ thay thế

</script>

### **Phương thức entries()**

<script>

const fruits = ["Banana", "Orange", "Apple", "Mango"];

const f = fruits.entries();

for (let x of f) {

document.getElementById("demo").innerHTML += x + "<br>";

}

</script>

0,Banana  
1,Orange  
2,Apple  
3,Mango

### **Phương thức every()**

const ages = [32, 33, 16, 40];  
  
ages.every(checkAge)    // Returns false  
  
function checkAge(age) {  
  return age > 18;  
}

***Syntax***

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

### **Phương thức fill()**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.fill("Kiwi");//["Kiwi", "Kiwi", "Kiwi", "Kiwi"];

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.fill("Kiwi", 2, 4);// ["Banana", "Orange", "Kiwi", "Kiwi"];

***Syntax***: *array*.fill(value, start, end)

### **Phương thức filter()**

const ages = [32, 33, 16, 40];  
  
ages.filter(checkAdult)    // Returns [32, 33, 40]  
  
function checkAdult(age) {  
  return age >= 18;  
}

<p><input type="number" id="ageToCheck" value="18"></p>  
  
<button onclick="myFunction()">Try it</button>  
  
<p id="demo"></p>  
  
<script>  
const ages = [32, 33, 12, 40];  
  
function checkAge(age) {  
  return age > document.getElementById("ageToCheck").value;  
}  
  
function myFunction() {  
  document.getElementById("demo").innerHTML = ages.filter(checkAge);

//32, 33, 40  
}  
</script>

### **Phương thức find()**

* Giống filter nhưng trả về giá trị đầu tiên phù hợp

const ages = [3, 10, 18, 20];  
  
function checkAge(age) {  
  return age > 12;  
}  
  
function myFunction() {  
  document.getElementById("demo").innerHTML = ages.find(checkAge);

// 18  
}

### **Phương thức findIndex()**

* Trả về vị trí đầu tiên thoã mãn

const ages = [3, 10, 18, 20];  
  
ages.findIndex(checkAge)   // Returns 2  
  
function checkAge(age) {  
  return age > 10;  
}

### **Phương thức foreach()**

let sum = 0;  
const numbers = [65, 44, 12, 4];  
numbers.forEach(myFunction);  
  
function myFunction(item) {  
  sum += item;  
}

<script>

const numbers = [65, 44, 12, 4];

numbers.forEach(myFunction)

document.getElementById("demo").innerHTML = numbers;//[650, 440, 120, 40]

function myFunction(item, index, arr) {

arr[index] = item \* 10;

}

</script>

### **Phương thức from()**

<script>

const myArr = Array.from("ABCDEAG");

document.getElementById("demo").innerHTML = myArr;// A,B,C,D,E,A,G

</script>

**Syntax**: Array.from(object, mapFunction, thisValue)

### **Phương thức includes()**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.includes("Banana", 3);   // Returns false

### **Phương thức indexOf()**

const fruits = ["Banana", "Orange", "Apple", "Mango", "Apple"];  
fruits.indexOf("Apple", 4)   // Returns 4

### **Phương thức isArray()**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
Array.isArray(fruits)   // Returns true

### **Phương thức keys()**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
const keys = fruits.keys();  
  
for (let x of keys) {  
  text += x + "<br>";

/\*

0

1

2

3

\*/  
}

### **Phương thức lastIndexOf()**

const fruits = ["Orange", "Apple", "Mango", "Apple", "Banana", "Apple"];  
fruits.lastIndexOf("Apple", 4)   // Returns 3

### **Phương thức map()**

* Phương thức tạo mới mảng với kết quả của việc gọi hàm tính toán các phần tử trong mảng
* Không thực thi mảng rỗng
* Không thay đổi mảng cũ

const numbers = [4, 9, 16, 25];  
const newArr = numbers.map(Math.sqrt)//[2, 3, 4, 5]

const persons = [  
  {firstname : "Malcom", lastname: "Reynolds"},  
  {firstname : "Kaylee", lastname: "Frye"},  
  {firstname : "Jayne", lastname: "Cobb"}  
];  
  
persons.map(getFullName);  
  
function getFullName(item) {  
  return [item.firstname,item.lastname].join(" ");// Malcom Reynolds,…  
}

### **Phương thức reduce()**

* Hàm rút gọn cho mỗi giá trị trong mảng
* Trả về một số kết quả tính toán
* Không thực thi mảng rỗng
* Không thay đổi mảng ban đầu

**Syntax:**

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

total (yêu cầu) : giá trị khởi tạo, trả về

currentValue (yêu cầu): giá trị của phần tử hiện tại

currentIndex (không yêu cầu): chỉ số mảng

arr (không yêu cầu): đối tượng mảng mà phần tử hiện tại thuộc về

const numbers = [15.5, 2.3, 1.1, 4.7];  
document.getElementById("demo").innerHTML = numbers.reduce(getSum, 0);  
  
function getSum(total, num) {  
  return total + Math.round(num);// hàm round làm tròn số =>kết quả: 24  
}

const numbers = [175, 50, 25];  
  
document.getElementById("demo").innerHTML = numbers.reduce(myFunc);100  
  
function myFunc(total, num) {  
  return total - num;  
}

const numbers = [175, 50, 25];  
  
document.getElementById("demo").innerHTML = numbers.reduce(myFunc, 1);//-249  
  
function myFunc(total, num) {  
  return total - num;  
}

### **Phương thức reduceRight()**

* Ngược lại với reduce thì reduceRight() tính từ phải sang trái

const numbers = [175, 50, 25];  
  
document.getElementById("demo").innerHTML = numbers.reduceRight(myFunc);// -200  
  
function myFunc(total, num) {  
  return total - num;  
}

### **Phương thức reserve()**

* Phương thức đảo ngược mảng

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.reverse();// ["Mango", "Apple", "Orange", "Banana"]

### **Phương thức some()**

* Ngược lại với every() kiểm tra tất cả các phần tử thoã mãn some() chỉ cần một phần tử đúng

const ages = [3, 10, 18, 20];  
  
ages.some(checkAdult)   // Returns true  
  
function checkAdult(age) {  
  return age >= 18;  
}

### **Phương thức sort()**

* Thay đổi thứ tự mảng cũ

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.sort();

const points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return a-b});

const points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return b-a});

## **1.20 JS Date**

- Có 4 cách khởi tạo đối tượng Date():

new Date()  
new Date(year, month, day, hours, minutes, seconds, milliseconds)  
new Date(milliseconds)  
new Date(date string)

* Lấy các thành phần ra từ đối tượng Date()
* .getFullYear() : 2021
* .getMonth() + 1 : 10
* .getDate(): 15
* .getHours() (0-23): 12
* .getMinutes() (0-59): 56
* .getSeconds() (0-59): 43
* .getDay() (2-6): 5
* .getTime(): 1634278831457 **(tính từ 01-01-1970)**
* Date.now()

### **Date Set Method**

* setDate()
* setFullYear()
* setHours()
* setMilliseconds()
* setMinutes()
* setMonth()
* setSeconds()
* setTime()

## **1.21 JS Math**

Math.E        // returns Euler's number  
Math.PI       // returns PI  
Math.SQRT2    // returns the square root of 2  
Math.SQRT1\_2  // returns the square root of 1/2  
Math.LN2      // returns the natural logarithm of 2  
Math.LN10     // returns the natural logarithm of 10  
Math.LOG2E    // returns base 2 logarithm of E  
Math.LOG10E   // returns base 10 logarithm of E

Math.round() //số làm tròn gần nhất

Math.ceil() //số làm tròn số cao hơn Math.ceil(4.2);     // returns 5

Math.floor() //số làm tròn số thấp

Math.truc() // trả về phần nguyên

Math.sign() //trả về âm, dương hoặc NaN

Math.pow(x, y) //x^y

Math.abs(x) //căn x

Sin() cos() // trả về giá trị từ -1 đến 1

min() //Math.min(0, 150, 30, 20, -8, -200);  // returns -200

max() //Math.max(0, 150, 30, 20, -8, -200);  // returns 150

Math.random()

Math.log(1);    // returns 0

Math.log2(8);    // returns 3

Math.log10(1000);    // returns 3

Xem thêm: <https://www.w3schools.com/js/js_math.asp>

1. **JS Object**
2. **JS Functions**
3. **JS Classes**
4. **JS Async**

## **5.1 Trình tự các hàm**

* JS xử lý hàm khai báo cuối cùng sẽ gọi (nếu không có đk nào)

## **5.2 Callback**

* Một hàm được truyền như một đối số của hàm khác

function myDisplayer(some) {  
  document.getElementById("demo").innerHTML = some;  
}  
  
function myCalculator(num1, num2, myCallback) {  
  let sum = num1 + num2;  
  myCallback(sum);//10  
}  
  
myCalculator(5, 5, myDisplayer);

myCalculator: hàm số

myDisplayer: đối số

## **5.3 JS không đồng bộ**

### **setTimeout()**

setTimeout(myFunction, 3000);  
  
function myFunction() {  
  document.getElementById("demo").innerHTML = "I love You !!";//sau 3s hiện lên  
}

setTimeout(function() { myFunction("I love You !!!"); }, 3000);  
  
function myFunction(value) {  
  document.getElementById("demo").innerHTML = value;  
}

### **setIntervals()**

* Chỉ định một hàm được gọi lại sẽ được thực thi trong một khoảng thời gian

setInterval(myFunction, 1000);  
  
function myFunction() {  
  let d = new Date();  
  document.getElementById("demo").innerHTML=  
  d.getHours() + ":" +  
  d.getMinutes() + ":" +  
  d.getSeconds();  
}// mỗi 1s thì myFunction() sẽ được thực thi

### **Waiting for files**

* Trường hợp muống tải một tập tin ở bên ngoài, nhưng ko thể sử dụng trước khi nó tải đầy đủ

function myDisplayer(some) {//3  
  document.getElementById("demo").innerHTML = some;  
}  
  
function getFile(myCallback) {//2  
  let req = new XMLHttpRequest();  
  req.open('GET', "mycar.html");  
  req.onload = function() {  
    if (req.status == 200) {  
      myCallback(this.responseText);  
    } else {  
      myCallback("Error: " + req.status);  
    }  
  }  
  req.send();  
}  
  
getFile(myDisplayer);//1

myDisplayer: sử dụng callback

## **5.4 JS Promises**

- “Pro ducing code”: đoạn code mất nhiều time

- “Consuming code”: đoạn code phải đợi kết quả

- Promise là đối tượng để liên kết 2 đoạn code trên

### **Promise Syntax**

let myPromise = new Promise(function(myResolve, myReject) {  
// "Producing Code" (May take some time)  
  
  myResolve(); // when successful  
  myReject();  // when error  
});  
  
// "Consuming Code" (Must wait for a fulfilled Promise)  
myPromise.then(  
  function(value) { /\* code if successful \*/ },  
  function(error) { /\* code if some error \*/ }  
);

### **Thuộc tính trong Promise**

* Pending: đang xử lý => result là undefined
* Fulfilled: hoàn thành => result là value
* Rejected: từ chối => result là error object
* Hỗ trợ 2 thuộc tính: state và result

function myDisplayer(some) {  
  document.getElementById("demo").innerHTML = some;  
}  
  
let myPromise = new Promise(function(myResolve, myReject) {  
  let x = 0;  
  
// The producing code (this may take some time)  
  
  if (x == 0) {  
    myResolve("OK");  
  } else {  
    myReject("Error");  
  }  
});  
  
myPromise.then(  
  function(value) {myDisplayer(value);},  
  function(error) {myDisplayer(error);}  
);

### **Waiting for a Timeout, sử dụng Promise**

//Sử dụng callback

setTimeout(function() { myFunction("I love You !!!"); }, 3000);  
  
function myFunction(value) {  
  document.getElementById("demo").innerHTML = value;  
}

//Sử dụng Promise

let myPromise = new Promise(function(myResolve, myReject) {  
  setTimeout(function() { myResolve("I love You !!"); }, 3000);  
});  
  
myPromise.then(function(value) {  
  document.getElementById("demo").innerHTML = value;  
});

### **Waiting for file, sử dụng promise**

//Sử dụng callback

function getFile(myCallback) {  
  let req = new XMLHttpRequest();  
  req.open('GET', "mycar.html");  
  req.onload = function() {  
    if (req.status == 200) {  
      myCallback(req.responseText);  
    } else {  
      myCallback("Error: " + req.status);  
    }  
  }  
  req.send();  
}  
  
getFile(myDisplayer);

//Sử dụng promise

let myPromise = new Promise(function(myResolve, myReject) {  
  let req = new XMLHttpRequest();  
  req.open('GET', "mycar.htm");  
  req.onload = function() {  
    if (req.status == 200) {  
      myResolve(req.response);  
    } else {  
      myReject("File not Found");  
    }  
  };  
  req.send();  
});  
  
myPromise.then(  
  function(value) {myDisplayer(value);},  
  function(error) {myDisplayer(error);}  
);

## **5.5 JS Async/Await**

- Async làm hàm để trả về một Promise

- Await làm hàm để chờ một Promise

### **Async**

async function myFunction() {  
  return "Hello";  
}

async function myFunction() {  
  return Promise.resolve("Hello");  
}

<script>

function myDisplayer(some) {

document.getElementById("demo").innerHTML = some; //Hello

}

async function myFunction() {return "Hello";}

myFunction().then(

function(value) {myDisplayer(value);},

function(error) {myDisplayer(error);}

);</script>

### **Await Syntax**

* Từ khoá await được sử dụng trước một hàm làm cho hàm chờ một promise xử lý
* Từ khoá await được sử dụng bên trong một hàm async

let value = await promise;

Ví dụ không reject:

async function myDisplay() {  
  let myPromise = new Promise(function(resolve) {  
    resolve("I love You !!");  
  });  
  document.getElementById("demo").innerHTML = await myPromise;  
}  
  
myDisplay();

//Sử dụng waiting Timeout

async function myDisplay() {  
  let myPromise = new Promise(function(resolve) {  
    setTimeout(function() {resolve("I love You !!");}, 3000);  
  });  
  document.getElementById("demo").innerHTML = await myPromise;  
}  
  
myDisplay();

### **Waiting for a File**

async function getFile() {  
  let myPromise = new Promise(function(resolve) {  
    let req = new XMLHttpRequest();  
    req.open('GET', "mycar.html");  
    req.onload = function() {  
      if (req.status == 200) {  
        resolve(req.response);  
      } else {  
        resolve("File not Found");//404  
      }  
    };  
    req.send();  
  });  
  document.getElementById("demo").innerHTML = await myPromise;  
}  
  
getFile();

1. **JS HTML DOM**

Sự kiện HTML DOM cho phép đki sự kiện của elements

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

1. **JS Brower BOM**
2. **JS Web APIs**
3. **JS Ajax**
4. **JS Json**