**JAVASCRIPT ROADMAP**

**1 BIẾN**

### Khai báo và khởi tạo biến

JavaScript là một ngôn ngữ kiểu dữ liệu động không cần khai báo rõ ràng về kiểu dữ liệu của nó ;

Có ba từ khóa khi khai báo biến javascript là: **var**, **let** và **const**.

**var** varName1 = 10; // khai báo và khởi tạo biến với từ khóa var

**let** varName2; // khai báo và không khởi tạo giá trị ban đầu

varName2 = 10; // gán giá trị

**const** varName3 = 10; // khai báo và khởi tạo biến với từ khóa const

**Hoisting**khai báo các biến và hàm được đẩy lên đầu phạm vi của chúng trước khi code thực thi

console.log (**UserName**); // in ra: undefined

**var** **UserName** = "xuanthulab";

**Sự khác nhau giữa var và let:**

* var cho phép khai báo lại (nhiều lần), let thì biến chỉ được khai báo 1 lần (nếu cố tình khai báo lần tiếp theo sẽ bị lỗi).
* let thì phạm vi hiệu lực trong khối {} nó khai báo, var thì hiệu lực toàn cục
* Các biến đều được JS kéo lên đầu khai báo khi thực thi, nhưng khi đẩy lên đầu biến khai báo với var được khởi tạo giá trị **undefined** còn biến khai báo
* với let thì không được khởi tạo khi kéo lên

**Ghi chú** : người ta thường ưu tiên sử dụng **let** hơn là **var** trong các phiên bản JavaScript hiện đại. **let** giúp giảm rủi ro của việc sử dụng biến trước khi nó được khai báo và cung cấp một phạm vi scope rõ ràng hơn làm việc an toàn hơn.

|  |  |
| --- | --- |
| **2 KIỂU DỮ LIỆU**  Kiểu của biến là kiểu mà giá trị biến đó đang chứa *(*như một biến khai báo và gán luôn cho nó một số thì biến đó kiểu số) | **var** myvar = 42; // myvar khởi tạo là một số nguyên  myvar = "Hi JS" //myvar giờ lại là biến chuỗi  myvar = true; //giờ lại là biến logic |
| **Kiểu logic - Boolean** Biểu diễn logic đúng / sai - có 2 giá trị là true và false  **var** found = false;  Giá trị logic của 0, null, undefined, chuỗi rỗng là false. Các giá trị khác là true | **Kiểu số - Number**  ví dụ phông phân biệt số nguyên, số thực  **var** num = 42; // Một số không có phần thập phân  var price = 55.55;  document.write(price); |
| **Kiểu Null - rỗng** Biến có giá trị null thì nó không chứa đối tượng nào cả (rỗng)  **var** object = null; | **Kiểu symbol** Kiểu này mỗi giá trị là duy nhất và không thể bị sửa đổi**var** sym1 = **Symbol**(); |
| **Kiểu Undefined** Giá trị này tự động gán cho một biến khi biến đó khai báo. **let** x;  **if** (x === undefined) {  console.**log**("Biến x chưa khởi tạo");} | **Kiểu Object** Một đối tượng là một bộ sưu tập các cặp giá trị (key-value)  Const person = {firstName:"John", lastName:"Doe"}; |

|  |  |
| --- | --- |
| **// Biến toàn cục**  var globalVar = "Tôi là biến toàn cục";  function exampleFunction() {  var localVar = "Tôi là biến cục bộ";  console.log(globalVar); // truy cập mn  console.log(localVar); //truy cập pv kb  }  exampleFunction();  // console.log(localVar); // Lỗi: localVar is not defined | **// Biến cục bộ**  function exampleFunction() {  // Biến cục bộ  let localVar = "Tôi là biến cục bộ";  // chỉ sử dụng trong phạm vi của hàm  console.log(localVar);  }  // console.log(localVar); // Lỗi: localVar is not defined |

**3 TOÁN TỬ**

## **Toán tử typeof**

Toán tử typeof trả về kiểu dữ liệu cần kiểm tra của một biến, một giá trị.

**var** a = 1; console.**log**(**typeof**(a)); // trả về number

**var** b = "Hi"; console.**log**(**typeof**(b)); // trả về string

**var** c = true; console.**log**(**typeof**(c)); // trả về boolean

**var** d = null; console.**log**(**typeof**(d)); // trả về object

**Các toán tử gán trong Javascript**

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Ví dụ** | **Ý nghĩa** |
| = | x = y | gán giá trị y vào x |
| += | x += y | x = x + y; cộng thêm vào x giá trị y |
| -= | x -= y | x = x - y |
| \*= | x \*= y | x = x \* y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |
| ??= | x ??= y | Toán tử **??=** trong js là phép gán khi null. Nếu x bằng null thì giá trị y gán cho x, nếu x đã khác null thì không thay  đổi gì |

### Toán tử ?? loại bỏ null (null coalescing) trong javascript

Toán tử **??** trả về biểu thức bên trái nếu nó khác null (undefined), trả về biểu thức bên phải nếu nó là null

**var** ketqua = biểu\_thức\_trái ?? biểu\_thức\_phải;

## **Toán tử logic trong Javascript (JS)**

|  |  |
| --- | --- |
| **Toán tử** | **Diễn tả** |
| && | phép và (toán tử && trong javascript) trả về true nếu 2 số hạng là true: a && b |
| || | phép hoặc (toán tử hoặc trong javascript) trả về true nếu 1 trong 2 số hạng là true : a || b |
| ! | phủ định; !a trả về giá trị ngược với biểu thức a |

## **Các toán tử số học JS**

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Mô tả** | **Ví dụ** |
| + | phép cộng | 25 + 5 = 30 |
| - | phép trừ | 10 - 5 = 5 |
| \* | phép nhân | 2\*3 = 6 |
| / | phép chia | 20 / 2 = 10 |
| % | lấy phần dư của phép chia | 56 / 3 = 2 |
| ++ | Tăng thêm 1 | var a = 10; a ++; //giá trị a là 11 |
| -- | giảm đi 1 | var a = 10; a --; //giá trị a là 9 |

### Toán tử ba ngôi js - Toán tử điều kiện

variable = (condition) ? value1: value2;

Nhận giá trị value1 nếu điều kiện là true, nhận value2 nếu điều kiện false.

## **Toán tử với chuỗi trong JS**

Toán tử với chuỗi sử dụng nhiều là nối hai chuỗi lại với nhau, sử dụng toán tử + để nối. Lưu ý toán tử này có thể nối số vào chuỗi.

var mystring1 = "Học viết mã ";

var mystring2 = "JavaScript.";

document.write(mystring1 + mystring2);

## **Toán tử so sánh trong JavaScript (JS)**

Toán tử so sánh sử dụng trong các biểu thức về logic để so sánh bằng nhau, khác nhau. Nó trả về giá trị true false

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Diễn tả** | **Ví dụ** |
| == | so sánh bằng (và giá trị, không so sánh về kiểu dữ liệu) | **var** v1 = (5 == 10); // false  **var** v2 = 5; // số  **var** v3 = "5 "; // chuỗi  **var** v4 = (v2 == v3);// true |
| === | **=== trong js** so sánh giống nhau (true nếu cùng giá trị và kiểu dữ liệu) | **var** v1 = (5 === 10); // false  **var** v2 = 5; // kiểu số number  **var** v3 = "5 "; // chuỗi string  **var** v4 = (v2 === v3); // false, giống giá trị nhưng khác kiểu |
| != | so sánh khác giá trị (dấu khác trong javascript chỉ so sánh và giá trị không xét đến kiểu) | **var** v1 = (5 != 10); // true  **var** v2 = (5 != "5"); // false |
| !== | Khác giá trị và khác kiểu | **var** v1 = (5 !== 10); // false - khác giá trị nhưng cùng kiểu  **var** v2 = (5 !== "10"); // true |
| > | lớn hơn | 10 > 5 true |
| >= | lớn hơn hoặc bằng | 10 >= 5 true |
| < | nhỏ hơn | 10 < 5 false |
| <= | nhỏ hơn | 10 <= 5 false |

**4 ĐIỀU KIỆN VÒNG LẶP**

## **Lệnh if, if ... else**

**if** (expression) {

// thi hành các lênh ở đây (khối if) nếu true

}

**else** {

// thi hành các lệnh (khối else) nếu điều kiện false

}

## **Lệnh switch**

**switch** (expression) {

**case** n1:

//.. thi hành nếu expression bằng n1

**break**;

**case** n2:

//.. thi hành nếu expression bằng n2

**break**;

default:

//.. mặc định thi hành nếu expression không bằng giá trị nào ở trên

}

## **Vòng lặp for**

**for** (statement1; statement2; statement3) {

Khối lệnh thi hành

}

* statement1 : lệnh thi hành trước khi vòng lặp for bắt đầu
* statement2 : điều kiện kiểm tra trước mỗi lần thi hành khối lệnh for (true thì khối lệnh sẽ thi hành, false sẽ khối for sẽ không thi hành - thoát lặp)
* statement3 : thi hành sau mỗi lần một vòng hoàn thành

### for ... in

**for ... in** duyệt qua các key của đối tượng đếm được (không dùng cho mảng)

**for** (**let** yourvar **in** object) {

}

**let** ob = {key1: 1, key2: 2};

**for** (**let** k **in** ob) {

console.**log**(k);

}//In ra: key1, key2

**for ... of**

Lệnh **for ... of** duyệt qua các đối tượng.

**for** (**let** yourvar **in** objects) {

}

**let** a = ["A", "B", "C"];

**for** (**let** e **in** a) {

console.**log**(e);

}

//In ra A,B,C

## **Vòng lặp while**

Thi hành khối lệnh khi mà điều kiện kiểm tra vẫn là true

**while** (điều-kiện) {

//Khối lệnh

}

## **Vòng lặp do while**

Giống với vòng lặp while nhưng khối lệnh thi hành luôn mà không kiểm tra điều kiện trước, khi khối lệnh thi hành xong mới kiểm tra điều kiện để xem có lặp lại hay không

**do** {

//Khối lệnh

}

**while** (condition);

## **Lệnh continue và break**

Trong vòng lặp khi gặp continue; nó sẽ bỏ qua các lệnh còn lại và khởi tạo vòng lặp mới luôn. Còn nếu gặp break; thì bỏ qua các lệnh còn lại đồng thời thoát khỏi vòng lặp

### Duyệt qua các phần tử mảng bằng forEach

courses.**forEach**(**function**(item, index, array) {

console.**log**(item, index);

});

//HTML 0

//CSS 1

//JS 2

* currentValue: Giá trị của phần tử hiện tại đang được lặp.
* index: Chỉ mục (index) của phần tử hiện tại.
* array: Mảng đang được lặp qua.

let numbers = [1, 2, 3, 4, 5];

numbers.forEach(function(number, index) {

console.log(`Element at index ${index}: ${number}`);

});

Element at index 0: 1

Element at index 1: 2

Element at index 2: 3

Element at index 3: 4

Element at index 4: 5

**5 HÀM**

### Gọi hàm

Để gọi hàm viết lại tên hàm và các tham số truyền vào hàm, nhớ kết thúc bằng dấu ; Một hàm có thể được gọi bao nhiêu lần là tùy bạn.

**function** **myFunction**() {

**alert**("Calling a Function!");

}

**myFunction**();

## **Tham số hàm**

Các tham số của hàm bạn cần liệt kê sau tên hàm, mỗi tham số cách nhau bởi dấu ,

**functionName**(param1, param2, param3 = **default**) {

// các dòng code trong hàm

}

## **Giá trị trả về của hàm**

Hàm có thể tùy chọn có giá trị trả về hay không. Nếu có giá trị trả về hàm sử dụng lệnh return

Khối lệnh của hàm mà gặp đoạn lệnh return hàm sẽ dừng thi hành và trả về biểu thức của giá trị của return

**function** **myFunction**(a, b) {

**return** a \* b;

}

**var** x = **myFunction**(5, 6);

// Giá trị trả về được gán vào x

// x bằng 30

Hàm không có return một giá trị cụ thể nào, thì giá trị trả về của hàm là undefined

## **Hàm ẩn danh anonymous**

Chính là biểu thức hàm ở trên, nhưng trong phần khai báo bỏ tên đi (không tên).

//sau từ khóa function không có tên => thành ẩn danh

**var** a = **function** (guestname) {

**alert**('Xin chào ' + guestname);

}

**a**("Expression");

## **Biểu thức hàm chạy ngay lập tức IIFE**

biểu thức sau sẽ chạy luôn!

(**function** (a,b,c) {

**var** tong = a + b + c;

//...

**alert**(tong);

})(5,6,7);

**Hàm rút gọn =>**

Cú pháp cơ bản như sau

//(1) Không có tham số

() => { ... }

//(2) Có 1 tham số

param => { ... }

//(3) Dạng tổng quát

(param1, param2) => { ... }

|  |  |
| --- | --- |
|  |  |
| **let** x = **function**() {  console.**log**("Xin Chào");  } | **let** **x** = () => {  console.**log**("Xin Chào");  } |
| **let** x = **function**(a) {  console.**log**(a);  } | **let** **x** = a => {  console.**log**(a);  } |
| **let** x = **function**(a, b) {  **return** a + b;  } | **let** **x** = (a,b) => {  **return** a + b;  } |

**Call back**

Hiểu đơn giản thì hàm callback là một hàm sẽ được gọi bởi một hàm khác. Hiểu phức tạp hơn thì callback một hàm A được truyền vào hàm B thông qua các tham số của hàm B. Bên trong hàm B sẽ gọi đến hàm A để thực hiện một chức năng nào đó.

// Hàm thực hiện phép cộng

function add(a, b) {

return a + b;

}

// Hàm thực hiện phép nhân

function multiply(a, b) {

return a \* b;

}

// Hàm thực hiện một phép toán và gọi callback để xử lý kết quả

function performOperation(x, y, Callback) {

const result = Callback(x, y);

console.log("Kết quả của phép toán là:", result);

}

// Gọi hàm performOperation và truyền hàm add , multiply làm callback

performOperation(5, 3, add);

performOperation(5, 3, multiply);

function A(){

   // code

}

// Hàm B có một tham số callback

function B(callback){

    callback();

}

// Gọi hàm B và truyền tham số là hàm A

B(A);

<script>

setTimeout(myFunction, 3000);

function myFunction() {

document.getElementById("demo").innerHTML = "I love You !!";

}

</script>

**Closure**

Closure là một hàm được tạo ra từ bên trong một hàm khác, nó có thể sử dụng các biến toàn cục, biến cục bộ của hàm cha và biến cục bộ của chính nó.

function outerFunction() {

// Biến bên trong hàm ngoài cùng

let outerVariable = "Tôi là biến của hàm ngoài cùng";

// Hàm nội bộ (hàm bên trong)

function innerFunction() {

console.log(outerVariable);

}

// Gọi hàm nội bộ

innerFunction();

}

// Gọi hàm ngoài cùng

outerFunction();

**6 ALERT, PROMPT, CONFIRM , SETINTERVAL SETTIMEOUT**

## **Hộp thoại Alert**

Một hộp thoại thông báo thông tin cho người dùng. Khi hộp thoại hiện thị người dùng bấm vào nút OK để đóng lại.

**alert**("Hello\nHow are you?");

## **Hộp thoại Prompt**

Khi hộp thoại hiện thị, người dùng gõ dữ liệu yêu cầu sau đó nhấnnút OK hoặc Cancel để thi hành tiếp.

Nếu nhấn OK trả về giá trị nhập vào, nếu Cancel trả về giá trị null

Hàm prompt() có 2 tham số

**var** user = **prompt**("Nhập tên của bạn");

**alert**(user);

## **Hộp thoại Confirm**

Hộp thoại này dùng để người dùng xác nhận. Khi hộp thoại hiện thị nếu người dùng nhấn vào OK thì trả về true, nhấn vào Cancel sẽ trả về false

**var** result = **confirm**("Bạn có muốn rời trang?");

**if** (result == true) {

**alert**("Cám ơn đã truy cập!");

}

**else** {

**alert**("Cảm ơn đã ở lại!");

}

## **Hàm setInterval**

Hàm setInterval() sẽ gọi một hàm khác (hoặc một đoạn code) cứ sau một khoảng thời gian ấn định. Hàm này trả về ID của quá trình gọi này. Giá trị ID được dùng để ngắt quá trình gọi lặp lại này bằng hàm clearInterval(ID)

Cú pháp:

**var** intervalID = setInterval(func, delay);

func là hàm sẽ được gọi sau mỗi khoảng thời gian delay (tính theo đơn vị mili giây)

Ví dụ sau cứ 3 giây lại hiện popup có nội dung Hi, sau khi hiện thị 5 lần thì ngắt quá trình gọi lặp lại này

<script>

**var** i = 0;

**var** id = setInterval(myAlert, 3000); //Cứ 3s gọi hàm myAlert một lần

**function** **myAlert**() {

i++;

**alert**("Hi " + i);

**if** (i == 5)

clearInterval(id); //Ngắt lặp lại sau 5 lần

}</script

## **SetTimeOut**

**setTimeout** là một hàm trong JavaScript được sử dụng để đặt một hàm (hoặc một chuỗi hàm) để thực thi sau một khoảng thời gian nhất định

// Định nghĩa một hàm sẽ được thực thi sau một khoảng thời gian

function myFunction() {

console.log("Hàm này đã được thực thi sau một khoảng thời gian.");

}

// Sử dụng setTimeout để đặt hàm myFunction để thực thi sau 2000 milliseconds (2 giây)

setTimeout(myFunction, 2000);

**7 MẢNG**

**Cách 1**, tạo mảng với new Array()

**var** courses = **new** **Array**("HTML", "CSS", "JS");

**Cách 2**, tạo mảng với khai báo dữ liệu trong []

**var** courses = ["HTML", "CSS", "JS"];

### Số lượng phần tử trong mảng

Thuộc tính length cho biết số lượng phần tử trong mảng

**var** courses = ["HTML", "CSS", "JS"];

document.**write**(courses.length);

//Outputs 3

### Kết hợp 2 mảng

Để kết hợp 2 mảng thành một mảng, dùng phương thức concat

**var** c1 = ["HTML", "CSS"];

**var** c2 = ["JS", "C++"];

**var** courses = c1.**concat**(c2);

### Thêm một phần tử vào cuối bằng push

courses.**push**("GO");

### Thêm một phần tử vào đầu bằng unshift

courses.**unshift**("C++");

### Xóa phần tử cuối của mảng bằng pop

courses.**pop**();

### Xóa phần tử đầu của mảng bằng shift

courses.**shift**();

### Xóa các phần tử theo chỉ số splice

//Xóa 2 phần từ bắt đầu phần tử ở vị trí index

courses.**splice**(index, 2);

### Nhân bản một mảng với slice

**var** courses2 = courses.**slice**();

|  |  |
| --- | --- |
| **Methods** | **Description** |
| [charAt()](https://www.javatpoint.com/javascript-string-charat-method) | Nó cung cấp giá trị char có mặt tại chỉ mục được chỉ định. |
| [charCodeAt()](https://www.javatpoint.com/javascript-string-charcodeat-method) | Nó cung cấp giá trị Unicode của một ký tự có tại chỉ mục được chỉ định. |
| [concat()](https://www.javatpoint.com/javascript-string-concat-method) | Nó cung cấp một sự kết hợp của hai hoặc nhiều chuỗi. |
| [indexOf()](https://www.javatpoint.com/javascript-string-indexof-method) | Nó cung cấp vị trí của một giá trị char có trong chuỗi đã cho. |
| [lastIndexOf()](https://www.javatpoint.com/javascript-string-lastindexof-method) | Nó cung cấp vị trí của một giá trị char có trong chuỗi đã cho bằng cách tìm kiếm một  ký tự từ vị trí cuối cùng. |
| [search()](https://www.javatpoint.com/javascript-string-search-method) | Nó tìm kiếm một biểu thức chính quy được chỉ định trong một chuỗi nhất định và trả  về vị trí của nó nếu một kết quả khớp xảy ra. |
| [match()](https://www.javatpoint.com/javascript-string-match-method) | Nó tìm kiếm một biểu thức chính quy được chỉ định trong một chuỗi nhất định và trả  về biểu thức chính quy đó nếu xảy ra kết quả khớp. |
| [replace()](https://www.javatpoint.com/javascript-string-replace-method) | Nó thay thế một chuỗi nhất định bằng sự thay thế được chỉ định. |
| [substr()](https://www.javatpoint.com/javascript-string-substr-method) | Nó được sử dụng để tìm nạp một phần của chuỗi đã cho trên cơ sở vị trí và độ dài bắt  đầu được chỉ định. |
| [substring()](https://www.javatpoint.com/javascript-string-substring-method) | Nó được sử dụng để tìm nạp một phần của chuỗi đã cho trên cơ sở chỉ mục được chỉ  định. |
| [slice()](https://www.javatpoint.com/javascript-string-slice-method) | Nó được sử dụng để tìm nạp một phần của chuỗi đã cho. Nó cho phép chúng tôi chỉ định chỉ số dương cũng như âm. |
| [toLowerCase()](https://www.javatpoint.com/javascript-string-tolowercase-method) | Nó chuyển đổi chuỗi đã cho thành chữ thường. |
| [toLocaleLowerCase()](https://www.javatpoint.com/javascript-string-tolocalelowercase-method) | Nó chuyển đổi chuỗi đã cho thành chữ thường trên cơ sở ngôn ngữ hiện tại của máy  chủ. |
| [toUpperCase()](https://www.javatpoint.com/javascript-string-touppercase-method) | Nó chuyển đổi chuỗi đã cho thành chữ hoa. |
| [toLocaleUpperCase()](https://www.javatpoint.com/javascript-string-tolocaleuppercase-method) | Nó chuyển đổi chuỗi đã cho thành chữ hoa trên cơ sở ngôn ngữ hiện tại của máy chủ. |
| [toString()](https://www.javatpoint.com/javascript-string-tostring-method) | Nó cung cấp một chuỗi đại diện cho đối tượng cụ thể. |
| [valueOf()](https://www.javatpoint.com/javascript-string-valueof-method) | Nó cung cấp giá trị nguyên thủy của đối tượng chuỗi. |
| split() | Nó chia một chuỗi thành mảng chuỗi con, sau đó trả về mảng mới được tạo đó. |
| trim() | Nó cắt khoảng trắng từ bên trái và bên phải của chuỗi. |

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

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

**Set  
Set** là một tập hợp các giá trị không trùng lặp. Nó không giữ theo thứ tự và chỉ chấp nhận giá trị duy nhất.  
// Tạo một Set

let mySet = new Set();

// Thêm giá trị vào Set

mySet.add(1);

mySet.add(2);

mySet.add(3);

mySet.add(1); // Giá trị trùng lặp, nhưng Set chỉ giữ giá trị duy nhất

// Kiểm tra sự tồn tại của một giá trị trong Set

console.log(mySet.has(2)); // true

console.log(mySet.has(4)); // false

// Lấy kích thước của Set

console.log(mySet.size); // 3

// Lặp qua các giá trị trong Set

mySet.forEach(value => console.log(value));

* **Khởi tạo**: let set = new Set();
* **Thêm phần tử**: set.add(value);
* **Xóa phần tử**: set.delete(value);
* **Kiểm tra tồn tại giá trị**: set.has(value);
* **Đếm tổng số phần tử**: set.size;
* **Xóa toàn bộ phần tử**: set.clear();

**Map  
Map** là một tập hợp các cặp khóa-giá trị, nơi mà mỗi khóa và giá trị có thể là bất kỳ kiểu dữ liệu nào. Điều này giúp lưu trữ và truy xuất dữ liệu theo cặp khóa-giá trị

* Khởi tạo: let map = new Map()
* Thêm phần tử: map.set('Name', 'Nguyen Van Cuong');
* Xóa phần tử: map.delete("Name");
* Kiểm tra phần tử tồn tại: map.has('Name')
* Đếm tổng số phần tử: map.size
* Xóa toàn bộ phần tử: map.clear();let myMap = new Map();

// Thêm cặp khóa-giá trị vào Map

myMap.set('name', 'John');

myMap.set('age', 25);

// In giá trị của 'age' trước khi cập nhật

console.log("Giá trị của 'age' trước khi cập nhật:", myMap.get('age')); // 25

// Cập nhật giá trị của 'age'

myMap.set('age', 26);

// In giá trị của 'age' sau khi cập nhật

console.log("Giá trị của 'age' sau khi cập nhật:", myMap.get('age')); // 26

**map** có thể được sử dụng để ánh xạ (map) một mảng gốc thành một mảng mới, trong đó mỗi phần tử của mảng mới được tạo ra bằng cách ánh xạ từ phần tử tương ứng của mảng gốc thông qua một hàm ánh xạ (callback).  
// Mảng gốc

let numbers = [1, 2, 3, 4, 5];

let squaredNumbers = numbers.map(function (num) {

return num \* num;

});

// Hiển thị mảng mới

console.log(squaredNumbers); // [1, 4, 9, 16, 25]

## **8 MATH**

Đối tượng Math là định nghĩa sẵn trong JS, nó chứa các thuộc tính và phương thức cho phép thi hành một số tác vụ về toán học.

**Các thuộc tính của Math**

|  |  |
| --- | --- |
| **Thuộc tính** | **Diễn tả** |
| E | Hằng số Euler |
| PI | Hằng số PI |
| LN2 | Logarite tự nhiên của 2 |
| LN10 | Logarite tự nhiên của 10 |

Đối tượng Math luôn có sẵn để sử dụng ngay mà không cần tạo mới.

document.**write**(**Math**.PI);

//In ra 3.141592653589793

### Các phương thức của Math

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| abs(x) | giá trị tuyệt đối của x |
| acos(x) | tính acos, đơn vị góc radian |
| asin(x) | tính asin |
| atan(x) | tính atan |
| ceil(x) | làm tròn tăng |
| cos(x) | tính cos |
| exp(x) | Tính Ex |
| floor(x) | làm tròn giảm |
| log(x) | tính log cơ số E |
| max(x, y, z, ..., n) | lấy giá trị max |
| min(x, y, z, ..., n) | lấy giá trị min |
| pow(x, y) | tính x mũ y |
| random() | giá trị ngẫu nhiên từ 0 đến 1 |
| round(x) | làm tròn tới số nguyên gần nhất |
| sin(x) | tính sin |
| sqrt(x) | tính căn bậc 2 |
| tan(x) | tính tang |

Ví dụ tính căn bậc 2

**var** n = **prompt**("Nhập một số", "");

**var** answer = **Math**.**sqrt**(n);

**alert**("Căn bậc 2 của " + n + " là " + answer);

## **9 ĐỐI TƯỢNG DATE**

Đối tượng Date , để làm việc với ngày tháng, nó chứa các phương thức để lấy ngày, giờ,

**var** d = **new** **Date**();

//Sử dụng thời gian Unix

**new** **Date**(milliseconds);

//Sử dụng chuỗi biểu diễn ngày tháng

**new** **Date**(dateString);

//Sử dụng các tham số năm, tháng, ngày, giờ, phút, giây, mini giây

**new** **Date**(year, month, day, hours, minutes, seconds, milliseconds);

Ví dụ

//Fri Jan 02 1970 00:00:00

**var** d1 = **new** **Date**(86400000);

//Fri Jan 02 2015 10:42:00

**var** d2 = **new** **Date**("January 2, 2015 10:42:00");

//Sat Jun 11 1988 11:42:00

**var** d3 = **new** **Date**(88,5,11,11,42,0,0);

Lưu ý JavaScript đếm tháng từ 0 đến 11. Tháng 1 là 0.

## **Các phương thức của Date**

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| getDate() | Lấy ngày của tháng (1 - 31) |
| getDay() | Ngày tuần (0-6) |
| getFullYear() | Lấy năm bốn số (yyyy) |
| getHours() | Lấy giờ (0-23) |
| getMilliseconds() | Mili giây(0-999) |
| getMinutes() | Lấy phút (0-59) |
| getMonth() | Lấy tháng (0-11) |
| getSeconds() | Lấy giây (0-59) |
| getTime() | Lấy thời gian Unix |

Vị dụ đồng hồ thời gian hiện tại

**function** **printTime**() {

**var** d = **new** **Date**();

**var** hours = d.**getHours**();

**var** mins = d.**getMinutes**();

**var** secs = d.**getSeconds**();

document.body.innerHTML = hours+":"+mins+":"+secs;

}

setInterval(printTime, 1000);

**10 ĐỐI TƯỢNG , CLASS**

## **Các đối tượng - Object**

một đối tượng là một danh sách các giá trị được viết theo cặp nameobject:value

Ví dụ sau tạo ra đối tượng lưu trong biến person :

**var** person = {

name: "John",

age: 31,

favColor: "green",

height: 183

};

Các thuộc tính của đối tượng trên là name, age, favColor, height ...

## **Truy cập đến thuộc tính đối tượng**

Có 2 cách để truy cập đến thuộc tính đối tượng để lấy giá trị thuộc tính hoặc gán giá trị vào thuộc tính, ví dụ:

objectName.propertyName //Cách 1

objectName['propertyName'] //Cách 2

**var** person = {

name: "John", age: 31,

favColor: "green", height: 183

};

**var** x = person.age;

**var** y = person['age'];

## **Khởi tạo đối tượng**

### Cách 1) Khởi tạo cố định

Cách thứ nhất là tạo bằng phương pháp cố định, là cách đã thực hiện ở trên (*tạo đối tường và khởi tạo luôn các thuộc tính cần có*)

**var** person = {

name: "John",

age: 42,

favColor: "green"

};

### Cách 2) Khởi tạo bằng hàm tạo

Với cách thứ 2 này bạn khai báo một hàm gọi là hàm tạo rồi tạo ra đối tượng bằng cú pháp new hamtao()

Trong hàm tạo hoặc các hàm thuộc đối tượng, sử dụng từ khóa this để tham khảo đến đối tượng, thông qua nó truy cập các thuộc tính *(ý nghĩa của this giống this trong Java, Php, C# ...)*

**function** **person**(name, age, color) { // Hàm khởi tạo

this.name = name; // this tham khảo đến đối tượng cần tạo

this.age = age;

this.favColor = color;

}

**var** p1 = **new** **person**("John", 42, "green"); // tạo đối tượng

**var** p2 = **new** **person**("Amy", 21, "red"); // tạo đối tượng

document.**write**(p1.age); // Outputs 42

document.**write**(p2.name); // Outputs "Amy"

## **Phương thức trong đối tượng**

Một đối tượng ngoài các thuộc tính ra nó còn chứa hàm gọi là phương thức, ví dụ truy cập một hàm

objectName.**methodName**()

Như bạn đã biết khi viết chuỗi bạn có sử dụng đến document.write() thì write chính là phương thức của đối tượng document

Phương thức thuộc về một đối tượng, đối tượng này được hàm tham khảo qua từ khóa this

Thường các phương thức được định nghĩa qua một hàm khởi tạo đối tượng.

//Hàm khởi tạo đối tượng

**function** **person**(name, age) {

this.name = name;

this.age = age;

this.changeName = **function** (name) {

this.name = name;

}

}

//Tạo đối tượng

**var** p = **new** **person**("David", 21);

p.**changeName**("John");

//Giờ p.name bằng "John"

Các phương thức bạn cũng có thể định nghĩa bên ngoài hàm khởi tạo, ví dụ

**function** **person**(name, age) {

this.name= name;

this.age = age;

this.yearOfBirth = bornYear; //Gán phương thức bên ngoài

}

//Hàm bên ngoài hàm tạo, hàm này được gán vào đối tượng qua hàm tạo ở trên

**function** **bornYear**() {

**return** 2016 - this.age;

}

**var** p = **new** **person**("A", 22);

document.**write**(p.**yearOfBirth**());

// Outputs 1994

## **Setter và Getter**

Một thuộc tính của của đối tượng còn thiết lập nó là hàm setter hoặc getter, nếu là setter nó chỉ được gọi qua toán tử gán giá trị cho nó, nếu là getter thì chỉ được gọi khi truy cập lấy giá trị thuộc tính.

Hàm setter định nghĩa bằng cách cho thêm set, hàm getter định nghĩa bằng cách cho thêm get

**var** obj = {

age: 0,

**set** **ageInfo**(age) { //Định nghĩa setter

console.**log**('setter - ' + age);

this.age = age;

},

**get** **ageInfo**() { //Định nghĩa getter

console.**log**('getter');

**return** "Thông tin tuổi: " + this.age;

}

};

obj.ageInfo = 25; //Gán -> Tự động gọi settter

**alert**(obj.ageInfo); //Không phải gán -> Tự động gọi getter

### Tạo Setter/Getter trong hàm tạo

Trong trường hợp muốn định nghĩa setter / getter trong hàm tạo đối tượng

**Object**.**defineProperty**(ob, 'namepro', {

set: **function**(x) {

//code setter ở đây

},

get: **function**() {

//code getter ở đây

}});

## **Lớp (Class) trong JS**

Lớp để tạo ra các đối tượng (thực chất là một cách thức khác tạo ra các [đối tượng Javascript](https://xuanthulab.net/doi-tuong-trong-javascript.html) đã biết). Đối tượng trong nó có các thuộc tính, phương thức. Tạo ra đối tượng sử dụng từ khóa class và hàm khởi tạo constructor

//Khai báo một lớp có tên Product

**class** **Product** {

//Hàm khởi tạo

**constructor**(name, price) {

this.name = name;

this.price = price;

this.infomation = `${name} - ${price}`;

}

//Khai báo một phương thức

**checkStore**(storeid) {

console.**log**(this.name + ' in store ' + storeid);

}

//Hàm getter

**get** **info**() {

**return** this.infomation;

}

//Hàm setter

**set** **info**(i) {

this.infomation = i;

}

//Phương thức tĩnh

**static** **convertMoney**(m) {

console.**log**(m);

**return** m + ' đồng';

}

}

//SỬ DỤNG LỚP

//Tạo một đối tượng từ lớp bằng new

**let** sanpham = **new** **Product**('Iphone', 1000);

//truy cập thuộc tính đối tượng sanpham.name

console.**log**(sanpham.name);

//gọi một phương thức của đối tượng

sanpham.**checkStore**(100);

//Gọi setter

sanpham.info = 'Thông tin sản phẩm ...';

//Gọi getter

console.**log**(sanpham.info);

//Gọi một hàm tĩnh

**Product**.**convertMoney**(100000);

Từ đoạn mã trên lưu ý mấy vấn đề

* Lớp khai báo với từ khóa class, bắt buộc có hàm khởi tạo constructor, hàm này được gọi khi khởi tạo đối tượng lớp new Product('Iphone', 1000)
* Các phương thức (checkStore) khai báo trong lớp giống như khai báo hàm nhưng bỏ đi khóa function
* Ở trước phương thức có từ khóa get thì nó gọi làm phương thức getter, nó được gọi như cách truy cập thông tin thuộc tính
* Ở trước phương thức có từ khóa set thì nó là setter, nó được gọi khi bị gán (=)
* Phương thức tĩnh có keyword static, được gọi qua tên lớp (nó không thể tham khảo đến đối tượng lớp qua this)

## **Tính kế thừa của lớp**

Giống như nhiều ngôn ngữ lập trình hướng đối tượng, JS cung cấp tạo ra một lớp mới có kế thừa lại thuộc tính, phương thức của một lớp khác. Để tạo ra một lớp con kế từ lớp cha dùng từ khóa extends

**class** **Computer** **extends** **Product** {

**constructor**(name, price, store) {

super(name, price);

this.store = store;

}

**set** **info**(i) {

//super.info(i) - nếu muốn thi hành phương thức của lớp cha

this.infomation = name + ':'+i;

}

**totalInStore**() {

console.**log**('totalInStore');

}

}

//Sử dụng

**let** sanpham = **new** **Computer**('Dell', 2000);

console.**log**(sanpham.name);

sanpham.**checkStore**(200);

sanpham.info = 'Thông tin sản phẩm ...';

console.**log**(sanpham.info);

sanpham.**totalInStore**();

Ở ví dụ trên bạn đã tạo ra một lớp mới có tên Computer kế thừa lớp cha là Product, do đó lóp con có các thuộc tính, phương thức như lớp cha và khai báo thêm những thuộc tính phương thức riêng.

* Nếu trong lớp con khai báo một phương thức trùng tên đã có ở lớp cha thì phương thức lớp con được sử dụng mặc định, nếu muốn gọi đến phương thức của lớp cha dùng dùng từ khóa super thay cho this để tham khảo đến lớp cha
* Hàm khởi tạo lớp cha có tham số, do vậy bắt buộc phải gọi hàm tạo của lớp cha bằng super(..); trong constructor của lớp con

## **11 DOM Chọn phần tử trong DOM**

Tất cả các phần tử HTML như body, a, p ... đều là đối tượng trong JavaScript. Đối tượng document có các phương thức giúp bạn chọn ra phần tử mong muốn. Nó có ba phương thức thông dụng:

<script>

//tìm 1 phần tử theo id của nó - trả về HTMLElement hoặc null nếu không thấy

document.**getElementById**(id)

//tìm các phần tử theo class - trả về HTMLCollection

document.**getElementsByClassName**(name)

//tìm các phần tử theo tên thẻ - trả về HTMLCollection

document.**getElementsByTagName**(name)

</script>

### getElementById

Ví dụ sau, tìm phần tử có id là demo, sau đó đổi nội dung phần tử thành Hello World!

<html>

<**head**>

<**title**>DEMO JAVASCRIPT</**title**>

</**head**>

<**body**>

<**div** id="demo">Demo select Element</**div**>

<**script**>

**var** ele = document.**getElementById**('demo');

ele.innerHTML = "Hello World!";

</**script**>

</**body**>

</html>

### getElementsByTagName

Phương thức getElementsByTagName tìm tất cả các phần tử có tên thẻ trong tham số hàm.

<html>

<**head**>

<**title**>DEMO JAVASCRIPT</**title**>

</**head**>

<**body**>

<**p**>hi</**p**>

<**p**>hello</**p**>

<**p**>hi</**p**>

<**script**>

**var** arr = document.**getElementsByTagName**("p");

**for** (**var** x = 0; x < arr.length; x++) {

arr[x].innerHTML = "Hello World!";

}

</**script**>

</**body**>

</html>

**getElementsByClassName**

Phương thức getElementsByClassName tìm tất cả các phần tử có thuộc tính class (lớp) đưa ra. Nó cũng trả về HTMLCollection

**var** arr = document.**getElementsByClassName**("demo");

//truy cập phần tử thứ 2 tìm được

arr[1].innerHTML = "Hi";

## **Phần tử trong DOM - HTMLElement**

Mỗi một phần tử trong DOM đều có các thuộc tính và phương thức cung cấp thông tin về phần tử đó trong DOM

Các thuộc tính của phần tử trong DOM:

* element.childNodes : trả về mảng các phần tử con của nó
* element.firstChild : phần tử con đầu tiên
* element.lastChild : phần tử con cuối cùng
* element.hasChildNodes : trả về true nếu có phần tử con, ngược lại là false
* element.nextSibling : phần tử cùng cấp tiếp theo
* element.previousSibling : phần tử cùng cấp trước
* element.parentNode : trả về nút cha

## **Thay đổi thuộc tính phần tử trong DOM**

thay đổi nội dung bằng thuộc tính innerHTML

Ví dụ: phần tử HTML <img> như đã biết có thuộc tính src để chỉ ra URL hình ảnh mà thẻ đó hiện thị, đã biết nó có thuộc tính với tên là src thì sau khi có phần từ này từ DOM, bạn có thể đọc, gán thuộc tính và nó sẽ cập nhật lại trang HTML nếu gán

<**img** id="myimg" src="orange.png" alt="" />

<**script**>

**var** el = document.**getElementById**("myimg");

el.src = "apple.png";

</**script**>

Tương tự, thay đổi thuộc tính href trong phần tử liên kết <a>

<**a** href="http://www.example.com">Some link</**a**>

<**script**>

**var** el = document.**getElementsByTagName**("a");

el[0].href = "https://xuanthulab.net";

</**script**>

## 

## **Đổi style phần tử (CSS)**

style của các phần tử HTML có thể được thay đổi thông qua JavaScript, để thay đổi truy cập vào thuộc tính style và thay đổi các thông số yêu cầu:

<div id="demo" style="width:200px">Nội dung Ví dụ</div>

<**script**>

**var** x = document.**getElementById**("demo");

x.style.color = "6600FF";

x.style.width = "100px";

</**script**>

Ví dụ trên đã thay đổi color và width của phần tử thẻ div

Ví dụ sau tìm tất cả phần tử <p> sau đó đổi màu sang màu đỏ, thay cỡ chữ là 30px.

<**button** onclick="changePcolor()">Thay đổi thẻ P</**button**>

<**script**>

**function** **changePcolor**() {

**var** plements = document.**getElementsByTagName**('p');

**for** (**var** i = 0; i <= plements.length - 1; i++) {

**var** element = plements[i];

console.**log**(element);

element.style.color = "red";

element.style.fontSize = "30px";}

}

</**script**>

# **Tạo mới thêm xóa phần tử Dom - JavaScript**

Tạo ra một nút phần tử HTML trong DOM, chèn phần tử HTML, xóa phần tử và thay thế phần tử trong DOM HTML

## **Tạo phần tử**

Bạn có thể dùng JavaScript để tạo mới phần từ HTML sau đó chèn vào trang web.

|  |  |
| --- | --- |
| document.createElement(tag\_name) | tạo ra phần tử có thẻ tag\_name như a, p, div ... |
| element.cloneNode() | Tạo ra một phần tử bằng cách nhân bản phần tử chỉ ra (element) |
| document.createTextNode(text) | tạo phần tử tử văn bản text HTML |

**var** node = document.**createTextNode**("Tạo ra một phần tử");

**var** linknode = document.**createElement**("a");

linknode.href="https://xuanthulab.net/";

linknode.innerText="xuanthulab.net";

Ví dụ trên sẽ tạo ra một nút text, nhưng nó chưa hiện thị cho đến khi bạn gắn phần tử đó vào HTML document để nó là phần tử con của một phần tử nào đó

|  |  |
| --- | --- |
| element.appendChild(newNode) | Thêm phần tử newNode vào phần tử element  nó trở thành phần tử con sau cùng của element |
| element.insertBefore(newNode, node2) | Chèn phần tử newNode nằm phía trước node2 |
| element.replaceChild(newNode, oldNode) | Thay thế phần tử oldNode bằng phần tử newNode |

Ví dụ sau tạo ra một phần tử đoạn văn p sau đó chèn nó vào phần tử div đang có sẵn trong DOM HTML

<div id ="demo">nội dung ví dụ</div>

<**button** onclick="add\_child()">KẾT QUẢ</**button**>

<**script**>

**function** **add\_child**() {

//tạo phần tử p

**var** p = document.**createElement**("p");

//tạo phần tử text

**var** node = document.**createTextNode**("Some new text");

//gắn node vào p

p.**appendChild**(node);

//Thay đổi một số thuộc tính của p

p.**appendChild**(node);

p.style.backgroundColor = 'red';

p.style.padding = "10px";

p.style.color = "white";

**var** div = document.**getElementById**("demo");

//gắn p vào div

div.**appendChild**(p);

}</**script**>

**Loại bỏ phần tử**

Để loại bỏ phần tử HTML, bạn chọn phần tử cha rồi sử dụng phương thức removeChild(node)

<div id="demo">

<**p** id="p1">This is a paragraph.</**p**>

<**p** id="p2">This is another paragraph.</**p**>

</div>

<**script**>

**var** parent = document.**getElementById**("demo");

**var** child = document.**getElementById**("p1");

parent.**removeChild**(child);

</**script**>

Ví dụ trên sẽ xóa bỏ phần tử đoạn văn thứ nhất

Bạn có thể sử dụng thủ thuật lấy thuộc tính parentNode để bỏ qua bước tìm phần tử cha trong DOM: child.parentNode.removeChild(child);

<div id="demo">

<**p** id="p1">This is a paragraph.</**p**>

<**p** id="p2">This is another paragraph.</**p**>

</div>

<**script**>

**var** child = document.**getElementById**("p1");

child.parentNode.**removeChild**(child);

</**script**>

## **Thay thế phần tử**

Để thay thể một phần tử bằng một phần tử khác dùng cú pháp element.replaceChild(newNode, oldNode). Trong đó element là nút cha

<div id="demo">

<**p** id="p1">This is a paragraph.</**p**>

<**p** id="p2">This is another paragraph.</**p**>

</div>

<**script**>

**var** p = document.**createElement**("p");

p.innerText = 'Đây là đoạn văn mới tạo ra';

**var** parent = document.**getElementById**("demo");

**var** child = document.**getElementById**("p1");

parent.**replaceChild**(p, child);</**script**>

# **JavaScript Events**

Sự thay đổi trạng thái của một đối tượng được gọi là **Sự kiện ,** Khi mã [javascript](https://www.javatpoint.com/javascript-tutorial) được bao gồm trong [HTML](https://www.javatpoint.com/html-tutorial), js phản ứng qua các sự kiện này và cho phép thực thi. Quá trình phản ứng qua các sự kiện này được gọi là **Xử lý sự kiện**. Do đó, js xử lý các sự kiện HTML thông qua **Event Handlers**.

## **Mouse events:**

|  |  |  |
| --- | --- | --- |
| **Sự kiện đã thực hiện** | **Trình xử lý sự kiện** | **Sự miêu tả** |
| bấm | Onclick | Khi nhấp chuột vào một phần tử |
| Di chuột qua | Onmouseover | Khi con trỏ chuột đi qua phần tử |
| chuột ra | Onmouseout | Khi con trỏ chuột rời khỏi một phần tử |
| chuột xuống | Onmousedown | Khi nhấn nút chuột qua phần tử |
| chuột lên | onmouseup | Khi nút chuột được nhả qua phần tử |
| di chuyển chuột | onmousemove | Khi chuyển động chuột diễn ra. |

## **Keyboard events:**

|  |  |  |
| --- | --- | --- |
| **Sự kiện đã thực hiện** | **Trình xử lý sự kiện** | **Sự miêu tả** |
| Keydown &; Keyup | onkeydown &; onkeyup | Khi người dùng nhấn và sau đó thả  phím |

## **Form events:**

|  |  |  |
| --- | --- | --- |
| **Event Performed** | **Event Handler** | **Description** |
| focus | onfocus | Khi người dùng tập trung vào một yếu tố |
| submit | onsubmit | Khi người dùng gửi biểu mẫu |
| blur | onblur | Khi tiêu điểm cách xa một yếu tố biểu mẫu |
| change | onchange | Khi người dùng sửa đổi hoặc thay đổi giá trị của một phần  tử biểu mẫu |

## **Window/Document events**

|  |  |  |
| --- | --- | --- |
| **Event Performed** | **Event Handler** | **Description** |
| load | onload | Khi trình duyệt hoàn tất quá trình tải trang |
| unload | onunload | Khi khách truy cập rời khỏi trang web hiện tại, trình duyệt  sẽ dỡ nó |
| resize | onresize | Khi khách truy cập thay đổi kích thước cửa sổ của trình duyệt |

// Initial square position

let squareX = 50;

let squareY = 50;

// Draw the square at the specified position

function drawSquare() {

ctx.clearRect(0, 0, canvas.width, canvas.height);

ctx.fillStyle = 'blue';

ctx.fillRect(squareX, squareY, 50, 50);

}

// Handle keydown event to move the square

document.addEventListener('keydown', (event) => {

switch (event.key) {

case 'ArrowUp':

squareY -= 10;

break;

case 'ArrowDown':

squareY += 10;

break;

case 'ArrowLeft':

squareX -= 10;

break;

case 'ArrowRight':

squareX += 10;

break;

}

drawSquare();

});

// Handle mouse click event to move the square to the clicked position

canvas.addEventListener('mousedown', (event) => {

squareX = event.clientX - canvas.getBoundingClientRect().left - 25;

squareY = event.clientY - canvas.getBoundingClientRect().top - 25;

drawSquare();

});

drawSquare();

**Tài liệu tham khảo**

[Learn JavaScript Tutorial - javatpoint](https://www.javatpoint.com/javascript-tutorial)

[JavaScript Tutorial (w3schools.com)](https://www.w3schools.com/js/default.asp)  
[Học Javascript với 9 chuyên đề từ dễ đến khó (Miễn Phí) (freetuts.net)](https://freetuts.net/hoc-javascript)