JavaScript

# **Khai báo biến**

Để khai báo biến ta sẽ bắt đầu bằng từ khóa var (var là viết tắt của từ variable - nghĩa là biến). Khai báo biến có cú pháp như sau:

**var [dấu cách] [tên biến];**

Theo cú pháp trên, mình sẽ định nghĩa một biến có tên là fullName với dự định để lưu tên đầy đủ của mình vào đó.

var fullName;

Tiếp theo, ta có thể lưu thông tin vào biến fullName này:

var fullName; // khai báo biến

fullName = 'Sơn Đặng'; // gán giá trị

Các bạn chú ý có dấu nháy đơn '' bao ngoài chữ Sơn Đặng. Đó là cách để thể hiện dữ liệu dạng chuỗi (văn bản) trong Javascript.

Khi đoạn mã trên được chạy (thực thi) Javascript sẽ tạo biến với tên fullName và gán giá trị 'Sơn Đặng' cho biến này. Một vùng nhớ trong RAM của máy tính sẽ được sử dụng để phục vụ việc lưu trữ những giá trị của biến khi chương trình được thực thi.

Chuỗi 'Sơn Đặng' đã được lưu vào vùng nhớ tương ứng với biến fullName. Ta có thể truy cập tới chuỗi này qua tên biến:

var fullName;

fullName = 'Sơn Đặng';

alert(fullName); // hiển thị giá trị của biến

Để đơn giản và ngắn gọn, ta có thể kết hợp việc khai báo biến và gán giá trị cho biến thành một dòng:

var fullName = 'Sơn Đặng'; // khai báo và gán giá trị

alert(fullName);

Ta cũng có thể khai báo nhiều biến trong cùng một dòng cách nhau bởi dấu , như sau:

var fullName = 'Sơn Đặng', age = 18, workAt = 'F8';

Trông có vẻ ngắn gọn, tuy nhiên mình khuyên các bạn không nên dùng cách này. Khi cần khai báo nhiều biến hơn thì cách này trở nên rất khó đọc.  
  
Ta nên khai báo biến trên mỗi dòng khác nhau để dễ đọc hơn (nên dùng cách này):

var fullName = 'Sơn Đặng';

var age = 18;

var workAt = 'F8';

Một số cách khai báo biến trên nhiều dòng khác như sau:

var fullName = 'Sơn Đặng',

age = 18,

workAt = 'F8';

Thậm chí có cả phong cách sau:

var fullName = 'Sơn Đặng'

, age = 18

, workAt = 'F8';

Về mặt kỹ thuật thì tất cả các cách đều tương tự nhau. Vì vậy dùng cách nào là tùy theo sở thích của bạn.

Khi gán giá trị dạng số cho biến chúng ta không sử dụng dấu nháy đơn '' bao bọc bên ngoài. Như ví dụ trên thì age = 18 ta sẽ viết luôn là số 18 và không sử dụng dấu nháy.

**Quy tắc đặt tên**

1. Tên biến có thể bao gồm chữ cái, số, dấu gạch dưới ( \_ ) và kí tự đô la ( $ )
2. Tên biến không thể bắt đầu bằng số, phải bắt đầu bằng một chữ cái hoặc dấu gạch dưới hoặc dấu đô la
3. Tên biến phân biệt chữ hoa và chữ thường. Vì vậy tenbien và TenBien là 2 biến khác nhau
4. Tên biến không được (không thể) đặt trùng với các từ khóa của Javascript

Từ khóa là những từ được Javascript sử dụng để tạo nên những quy chuẩn về mặt chức năng và cú pháp trong Javascript. Ví dụ: Để khai báo một biến ta sẽ sử dụng từ khóa var. Vì vậy ta không thể đặt tên biến là "var"

**Ví dụ cho tên biến hợp lệ**

var address; // tên biến sử dụng chữ cái

var first\_name; // kết hợp chữ cái và gạch dưới

var $last\_name; // dấu đô la, gạch dưới và chữ cái

var f8; // chữ cái và số, số đứng sau chữ cái

**Ví dụ cho tên biến không hợp lệ**

var java-script; // bao gồm dấu gạch ngang

var 8f; // bắt đầu với chữ số

var var = 'Biến'; // sử dụng trùng từ khóa `var`

Các chữ cái không phải tiếng Lating vẫn có thể được sử dụng làm tên biến (không sử dụng cách này):

var ດ້ານວິຊາການ = '...'; // tiếng Pháp

var ਤਕਨੀਕੀ = '...'; // tiếng Lào

Trong thực tế chúng ta sẽ sử dụng tiếng Anh để đặt tên biến vì đó là quy ước chung Quốc Tế.

Gán giá trị cho biến

Các bạn hãy tưởng tượng biến như một chiếc hộp và giá trị gán cho biến như là đồ vật được bỏ vào hộp. Vì vậy ta có thể đặt bất cứ giá trị gì vào hộp và ta cũng có thể thay thế chúng nếu muốn:

var fullName; // tạo chiếc hộp

fullName = 'Sơn Đặng'; // cho đồ vật vào hộp

fullName = 'Nguyễn Văn A'; // thay thế đồ vật khác

alert(fullName); // Nguyễn Văn A

Khi giá trị của biến được thay đổi, giá trị cũ sẽ bị xóa khỏi biến.

Ta cũng có thể sao chép giá trị từ biến này sang biến khác:

var currentCourse = 'Javascript';

var newCourse;

// copy giá trị 'Javascript' từ biến 'currentCourse' sang biến 'newCourse'

newCourse = currentCourse;

// bây giờ, biến 'newCourse' và 'currentCourse'

// đều có giá trị là 'Javascript'

alert(currentCourse); // Javascript

alert(newCourse); // Javascript

Có thể bạn chưa biết có những ngôn ngữ lập trình như Scala, Erlang không cho phép thay đổi giá trị của biến đã định nghĩa. Ta bắt buộc phải tạo biến mới khi cần lưu giá trị và không thể gán lại giá trị cho biến cũ.

Đặt tên biến như nào cho đúng?

Đặt tên biến hợp lệ theo quy tắc của Javascript là việc đơn giản, tuy nhiên trong thực tế đặt tên biến không chỉ dừng lại ở việc đặt cho hợp lệ mà ta còn phải quan tâm tới các yếu tố khác như:

1. Tên biến phải có ý nghĩa cụ thể, phải rõ ràng và thể hiện được nó đang lưu trữ cái gì.
2. Sử dụng tiếng Anh để đặt tên biến, sử dụng các từ có thể đọc lên được như userName, phoneNumber, verifyEmail, ..
3. Tránh đặt tên biến ngắn như a, b, p trừ khi bạn chỉ đang làm ví dụ hoặc bạn thật sự hiểu trường hợp đó có thể đặt tên như vậy.
4. Tránh đặt tên biến chung chung kiểu như data, value. Vì khi nhìn vào không thể hiểu data là data của cái gì, value là value của cái gì. Chỉ sử dụng tên dạng này khi đang trong ngữ cảnh cụ thể giúp bổ nghĩa cho những từ chung chung đó.

Đặt tên biến chung chung (trường hợp nên tránh)

Ví dụ:

var data = '...'; // không biết data là data của cái gì

var value = '...'; // không biết value là value của cái gì

// var documentData = '...' ; Nên đặt rõ ràng ra như này

// var documentValue = '...'; và như này

Đặt tên biến chung chung (trường hợp nên dùng)

Ví dụ:

function Document() {

var data = '...';

// hoặc

var value = '...';

// var documentValue = '...'; Đặt như này sẽ bị lặp lại chữ "document" không cần thiết

}

Bạn chưa cần quan tâm function là gì vì ta sẽ học nó ở những bài sau. Trong trường hợp này biến data hoặc value nằm trong Document. Vì vậy Document đã giúp lập trình viên khi nhìn vào hiểu được data, value là thuộc về Document. Trong trường hợp này thì tên biến giúp đơn giản hóa và vẫn truyền đạt được đầy đủ ý nghĩa.

**Có thể bạn chưa biết**

1. Đặt tên biến là một trong những kỹ năng quan trọng và phức tạp nhất trong lập trình. Nhìn lướt qua các tên biến có thể biết code nào được viết bởi người mới và người đã có nhiều kinh nghiệm.
2. Trong thực tế nhiều khi chúng ta phải làm việc trên code đã có sẵn thay vì viết hoàn toàn mới. Có khi bạn sẽ làm việc trên code cũ của người khác và ngược lại. Vì vậy đặt tên biến rõ ràng, dễ hiểu, truyền đạt đúng mục đích sử dụng là quan trọng hơn cả.
3. Chỉ sau vài tháng bạn có thể quên đi đoạn mã do chính tay mình viết. Để chính bạn hiểu bạn đã từng code cái gì trong quá khứ thì việc đặt tên biến tuân thủ các nguyên tắc trên là vô cùng quan trọng.
4. Khi phải lựa chọn giữa performance (hiệu năng) và clean code (code sạch) người ta thường lựa chọn clean code. Việc đánh đổi này là cần thiết để giúp code dễ hiểu, dễ bảo trì và nâng cấp về sau. Và đặt tên biến chính là một trong những yếu tố giúp code của bạn trở nên clear hơn.

**Fact: Code cho máy hiểu thì dễ, code cho người hiểu mới khó!**

# **Comment**

**Mục đích sử dụng :**

Ghi chú

Vô hiệu hóa mã : Bản chất là trình biên dịch code sẽ bỏ qua những comment (không thực thi chúng). Vì vậy khi một đoạn code được comment lại thì đoạn code đó sẽ không chạy. Đôi khi bạn sẽ muốn tạm bỏ đi một đoạn code nào đó trong ứng dụng của bạn, đó chính là lúc bạn có thể sử dụng comment.

-Comment 1 dòng : //

-Comment nhiều dòng : /\* \*/

**Phím tắt :**

* Windows : Ctrl + /
* MacOS : Command + /

# **Giới thiệu một số hàm Built-in**

|  |  |
| --- | --- |
| Alert |  |
| Console | var a = 1;  console.log('Hello, world!');  console.log(123);  console.log(a);  console.warn('This is a warning'); // in ra màu vàng  console.error('This is an error'); //in ra màu đỏ  **Có thể truyền hai tham số** |
| confirm | cho người dùng xác nhận (cancel hoặc ok)  confirm('Are you sure?'); |
| prompt | cho người dùng nhập dữ liệu  window.prompt(“”Nhập tên : ) |
| setTimeout | truyền vào 1 function và thời gian chờ  setTimeout(function() {      console.log('Hello, world!');  }, 1000);  Dùng để chạy 1 hàm sau 1 khoảng thời gian nào đó |
| setInterval | truyền vào 1 function và thời gian chờ  setInterval(function() {      console.log('Hello, world!');  }, 1000);  Dùng để chạy 1 hàm lặp đi lặp lại sau 1 khoảng thời gian nào đó |
| Document.write | In ra trực tiếp trên trang web  document.write(“Xin chao”) |

# **Toán tử**

* **Toán tử số học :** + ; - ; \* ; / ; \*\* (lũy thừa) ; % (chia lấy dư) ; ++ (tăng 1 số) ; -- (giảm 1 số)
* **Toán tử gán :** var fullName = ‘Phan’
* **Toán tử so sánh : > ; < ; == ; >= ; <=**
* **Toán tử logic :** && ;

**Lưu ý :**

Toán tử ++ khi dùng là tiền tố sẽ tăng giá trị của biến lên 1 và trả về giá trị sau khi tăng.

Toán tử ++ khi dùng là hậu tố sẽ tăng giá trị của biến lên 1 và trả về giá trị trước khi tăng

**Tóm lại :**

* x++ tăng giá trị biến lên 1 và trả về giá trị trước khi tăng
* ++x tăng giá trị biến lên 1 và trả về giá trị sau khi tăng
* x-- giảm giá trị biến xuống 1 và trả về giá trị trước khi giảm
* --x giảm giá trị biến xuống 1 và trả về giá trị sau khi giảmm

Hãy xem thêm trong file word LeanJava

**Toán tử chuỗi - String operator**

Toán tử nối chuỗi là lấy 2 chuỗi + với nhau

‘Son’ + ‘Dang’ = SonDang

**Toán so sánh**

Tương tự Java : ===, > , < , != , <=, >=

Toán tử bằng tuyệt đối: ===

=> Bởi vì toán tử “==” chỉ so sánh value nên 1 (kiểu int) sẽ bằng với “1” (kiểu String)

Toán tử khác tuyệt đối: !== (Tương tự)

Ví dụ ‘A’ && ‘B’ && ‘C’ thì nó sẽ in ra ‘C’

**Toán tử logic**

Tương tự Java : && , ||, ! (đổi đúng thành sai, sai thành đúng, đặt phía trước)

=> Chỉ trả về kết quả của 1 trong hai vế . Chứ bản thân nó không trả về True hay False

**Toán tử Boolean**

var isSuccess = true;

var isSuccess = a > b;

**If -else**

if(bieu thuc dieu kien){

}else{

}

Trong javascript biểu thức điều kiện trong if tự động chuyển sang TRUE và FALSE

|  |  |
| --- | --- |
| **TRUE** | **FALSE** |
|  | 0 |
| flase |
| ‘’ hoặc “” |
| underfined |
| NaN |
| null |

**Kiểu dữ liệu trong JavaScript**

|  |  |  |
| --- | --- | --- |
| Kiều dữ liệu nguyên thủy  Giá trị tạo ra đi vào vùng nhớ và không thể sửa lại được giá trị nguyên thủy | Number |  |
| String |  |
| Boolean |  |
| Undefined | Tạo biến mà không gán giá trị  var age; |
| Null | var isNull = null  Không có gì |
| Symbol | var id = Symbol(‘id’)  Tính duy nhất  Ít dùng |
| Kiểu dữ liệu phức tạp (Complex Data) | Function (Hàm) | var myFunction = funtion(){  //code  }  Chỉ chạy khi được gọi  myFunction (); |
| Object | Có 2 loại :  + Object  var myObject = {  name : ‘Nguyen’,  age : 18,  };  +Array  Var myArray = [‘Javascript’, ‘PHP’, ‘Ruby’  ]; |

CPU xử lý

RAM lưu lại dữ liệu trong quá trình thực thi

**typeof tenBien : kiểm tra kiểu dữ liệu**

# **Truthy và Falsy là gì?**

**Truthy - to bool is true**

Bất cứ giá trị nào trong Javascript khi chuyển đổi sang kiểu dữ liệu boolean mà có giá trị true thì ta gọi giá trị đó là Truthy.

Các giá trị 1, ['BMW'], { name: 'Miu' } và 'hi' được đề cập trong ví dụ dưới đây là Truthy vì khi chuyển sang Boolean ta nhận được giá trị true.

!! là gì? Đơn giản thôi. Toán tử ! là toán tử not (phủ định) nên !! là 2 lần phủ định, mà 2 lần phủ định lại trở thành "khẳng định". Trong Javascript thì đây là một "tip" để convert (chuyển đổi) mọi kiểu dữ liệu khác sang Boolean.

**Falsy - to bool is false**

Bất cứ giá trị nào trong Javascript khi chuyển đổi sang kiểu dữ liệu boolean mà có giá trị false thì ta gọi giá trị đó là Falsy.

Trong Javascript có 6 giá trị sau được coi là Falsy:

- false, 0 (số không), chuỗi rỗng, null, undefined, NaN

**Ngoại lệ? - document.all**

Trong Javascript (phía trình duyệt) sẽ có sẵn một đối tượng document, và khi bạn thử !!document.all sẽ trả về false. Chẳng lẽ document.all cũng là falsy hay sao?

Bản thân mình cũng thắc mắc điều này nên mình đã search Google "Why document.all is falsy?"

Tóm tắt câu trả lời:

document.all là một ngoại lệ chính thức duy nhất theo đặc tả ECMA (phiên bản 5). Đặc tả này mô tả toàn bộ các object khi chuyển sang boolean sẽ là true. Tuy nhiên, document.all là một ngoại lệ.

Cụ thể như sau:

document.all chuyển sang boolean sẽ là false

document.all khi là toán hạng của toán tử so sánh == hoặc != sẽ là undefined

Khi typeof document.all sẽ trả về "undefined"

ECMA là đặc tả chi tiết kỹ thuật mà các ngôn ngữ theo đặc tả này phải tuân theo. Javascript là một ngôn ngữ tuân thủ đặc tả kỹ thuật ECMA.

**Hàm (function)**

-Là một khối mã

-Làm một việc cụ thẻ

-Chỉ thực thi khi được gọi

-Có thể nhận tham số và trả về giá trị

-Hàm trong javascript không cần truyền vào kiểu dữ liệu tham số và kiểu dữ liệu trả về.

**() gọi là toán tử call**

**Giá trị truyền vào function gọi là đối số**

**-**Kiểu dữ liệu tham số không giới hạn

-Biến chọn làm tham số có tình private (riêng tư)

-Số lượng tham số không giới hạn. Cách nhau bằng dấu “,”

-Nếu hàm bắt truyền 2 tham số mà mình chỉ truyền 1 tham số thì vẫn bình thường. Nó sự chuyển tham số bị thiếu thành “underfine”

* 1. **Một số điều lưu ý về function:**
* Khi function đặt trùng tên thì máy sẽ lấy hàm cuối cùng để thực thi
* Có thể khai báo biến trong hàm (tương tự Java) : biến được định nghĩa trong hàm thì phạm vi sử dụng chỉ ở trong hàm (function) thôi
* Có thể định nghĩa hàm (function) trong hàm (function)
  1. **Các loại function**

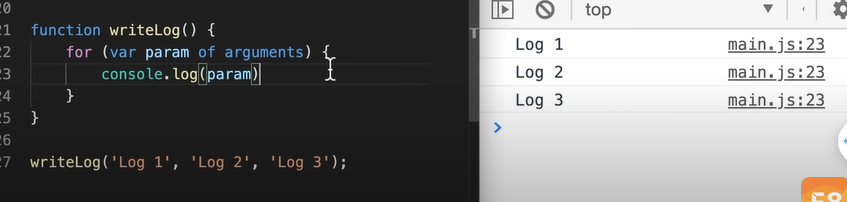
|  |  |
| --- | --- |
| **Declaration function** | **function ten(){**  **}** |
| **Expression function** | **var ten = function(){**  **}**  **var myObject = {**  **muFunction : function(){}**  **}**  **setTimeout(function(){**  **});** |
| **Arrow function** |  |
| Hosting: có thể giúp cho function được gọi trước khi được định nghĩa (Declaration function)  Tuy nhiên Expression function thì không có cơ chế hosting. | |

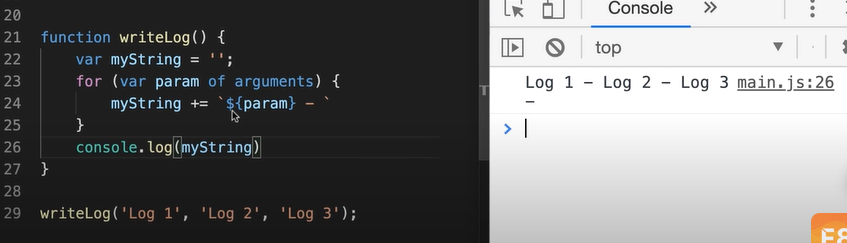
**Đối tượng Arguments ?**

-Chỉ xuất hiện trong funtion

-Giúp chúng ta truyền bao nhiêu biến cũng được

**Vòng for of**





**Return**

Nếu không return thì trả về underfined

Nếu có return thì toàn bộ các đoạn code sau return sẽ không thực thi.

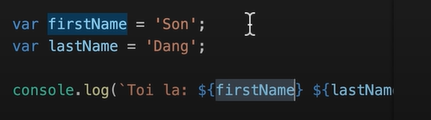
Return có thể trả về bất cứ dữ liệu gì

# **Chuỗi**

* + 1. **Tạo chuỗi**
    2. **Một số case sử dụng backslash (\)**
    3. **Xem độ dài chuỗi**

Tên chuỗi.length

* + 1. **Chú ý độ dài khi viết code (một dòng chỉ 80 kí tự)**
    2. **Template string ES6**

****

# **Làm việc với chuỗi**

Keyword : JS string methods

|  |  |
| --- | --- |
| Length | myString.length |
| Find index (tìm vị trí xuất hiện đầu tiên -> trả về int) | myString.indexOf(‘JS’) |
| myString.indexOf(‘JS’,5) : tìm sau vị trí số 5 |
| myString.lastIndexOf(‘JS’) : tìm vị trí cuối cùng của chuỗi cần tìm |
| myString.search(‘JS’) : giống indexOf, tìm theo biểu thức chính quy |
| Cut string | myString.slice(vitribatdau, vitriketthuc); |
| Replace (ghi đè) | myString.replace(‘chuMuonTim’, ‘chuMuonThay’) : chỉ sửa 1 chữ đầu tiên |
| myString.replace(‘/chuMuonTim/g’, ‘chuMuonThay’) : dùng biểu thức chính quy |
| Convert to upper case | myString.toUpperCase() |
| Convert to lower case | myString.toLowerCase() |
| Trim (thừa khoảng trắng) | myString.trim() : loại bỏ kí tự khoảng trắng thừa ở 2 dầu |
| Split |  |
| Get a character by index | charAt() : lấy kí tư theo vị trí |

# **Number**

**Từ khóa JS number method**

|  |  |
| --- | --- |
| **Phương thức** | **Vai trò** |
| Number.isFinite() | Xác định xem giá trị đã cho có phải là số hữu hạn hay không. Trả về boolean |
| Number.isInteger() | Xác định xem giá trị đã cho có phải là số nguyên hay không. Trả về boolean |
| Number.parseFloat() | Chuyển đổi chuỗi đã cho thành một số dấu phẩy động |
| Number.parseInt() | Chuyển đổi chuỗi đã cho thành một số nguyên |
| Number.prototype.toFixed() | Chuyển đổi và trả về chuỗi đại diện cho số đã cho, có số chữ số chính xác sau dấu thập phân |
| isNaN(giaTri) | Kiểm tra giaTri có phải NaN hay không ? |

# **Array**

-Trong JS mảng có thể bỏ nhiều kiểu dữ liệu vào .

* + - 1. **Cách tạo mảng**

var languages = [‘A1’, ‘A2’]; -> Cách 1

var languages = new Array (‘A1’, ‘A2’); -> Cách 2 : lưu ý hạn chế sử dụng

* + - 1. **Kiểu dữ liệu**

Kiểu dữ liệu khi typeof là object

Array.isArray(giaTri) : kiểm tra giá trị có phải là Array hay không.

* + - 1. **Các hàm**

|  |  |
| --- | --- |
| toString |  |
| Nối tất cả các phần tử của mảng thành một chuỗi, với separator là ký tự ngăn cách giữa các phần tử | Biến Array thành chuỗi.  mang.join(‘,’) |
| Xóa element cuối mảng và trả về phần tử đã xóa | mang.pop() |
| Thêm một hoặc nhiều phần tử vào cuối mảng và trả về số phần tử sau khi thêm. | mang.push(‘tenMuonThem’) |
| Xóa element đầu mảng và trả về phần tử đã xóa | mang.shift() |
| Thêm 1 hoặc nhiều phần tử vào đầu mảng và trả về số phần tử sau khi thêm. | mang.unshift(‘tenMuonThem’) |
| Xóa phần tử từ viTri trở về sau bao nhiêu phần tử.  Có thể vừa xóa vừa chèn (chèn phía trước) | mang.splice(viTri,soPhanTuMuonXoa,coTheTenMuonChen) |
| Phương thức này tương tự như splice(), nhưng thay vì thay đổi mảng gốc, nó trả về một bản sao mới của mảng với các thay đổi đã áp dụng. | array.toSpliced(start, deleteCount, item1, item2, ...); |
| Nối 2 mảng | mang1.concat(mang2) |
| Thêm 1 hoặc nhiều phần tử vào mảng | mang.push() |
| Trả về phần tử tại vị trí chỉ định. Hỗ trợ chỉ mục âm để lấy phần tử từ cuối mảng. | at(index) |
| Trả về một mảng con từ vị trí start đến end (không bao gồm end). Mảng gốc không bị thay đổi. | slice(start, end) |
| Trả về vị trí đầu tiên mà phần tử element xuất hiện trong mảng. Trả về -1 nếu không tìm thấy. | indexOf(element) |
| Tương tự indexOf, nhưng tìm kiếm từ cuối mảng về đầu. | lastIndexOf(element) |
| Kiểm tra xem mảng có chứa phần tử element không. Trả về true hoặc false. | includes(element) |
| Sắp xếp các phần tử của mảng theo thứ tự tăng dần (mặc định theo chuỗi Unicode). Có thể truyền vào hàm so sánh compareFunction để tùy chỉnh cách sắp xếp. | sort([compareFunction]) |
| Đảo ngược thứ tự các phần tử trong mảng.0 | reverse() |
| Chia chuỗi thành mảng các chuỗi con, dựa trên ký tự separator. | split(separator) |
| Chuyển đổi mảng thành chuỗi, các phần tử ngăn cách bằng dấu phẩy. | toString() |
| Tương tự toString(), nhưng hiển thị các phần tử dựa trên ngôn ngữ địa phương. | toLocaleString() |
| phương thức của mảng dùng để làm phẳng một mảng nhiều chiều (mảng lồng nhau) thành một mảng một chiều. | array.flat([depth]);  **depth** (tùy chọn): Số nguyên xác định độ sâu của việc làm phẳng. Giá trị mặc định là 1. Nếu bạn đặt depth là Infinity, mảng sẽ được làm phẳng hoàn toàn. |
| Điền giá trị value vào các phần tử từ vị trí start đến end (không bao gồm end). Mảng gốc bị thay đổi | fill(value, start, end) |
| Sao chép một phần của mảng đến vị trí khác trong mảng, mà không thay đổi độ dài của mảng | copyWithin(target, start, end) |
| Trả về một đối tượng Array Iterator chứa các cặp [index, element] của mảng | entries() |
| Trả về một đối tượng Array Iterator chứa các chỉ số (index) của mảng. | keys() |
| Trả về một đối tượng Array Iterator chứa các giá trị của mảng. | values() |
| Tạo một mảng mới từ một đối tượng tương tự mảng (array-like object) hoặc một iterable. | from(arrayLike) |
| Tạo một mảng mới từ các phần tử đã truyền vào. | of(...elements) |

# **Object**

Có thể thêm key – value cho Object một cách dễ dàng

Nếu key không có trả về ‘underfine’

Để xóa 1 cặp key- value : delete tenObject.tenKey

Gọi key có value là function : tenObject.tenKey() : lúc nào function sẽ được gọi

## **Object Constructor**

|  |
| --- |
| Công dụng của **Object Constructor** trong JavaScript |
| **Tạo nhiều đối tượng có cấu trúc giống nhau**: Bạn có thể tạo ra nhiều đối tượng với các thuộc tính và phương thức giống nhau mà không cần lặp lại mã. |
| **Đóng gói logic**: Các phương thức được định nghĩa trong constructor có thể sử dụng dữ liệu của đối tượng, giúp đóng gói logic liên quan đến đối tượng trong một nơi. |
| **Tái sử dụng mã:** Bạn chỉ cần định nghĩa cấu trúc đối tượng một lần trong constructor và có thể sử dụng lại khi cần tạo các đối tượng mới, giúp mã dễ quản lý và bảo trì hơn. |
| **Tăng tính linh hoạt**: Bạn có thể dễ dàng tạo các đối tượng khác nhau bằng cách thay đổi các giá trị đầu vào của constructor. |

-Tạo nên một bảng thiết kế (tương tự xây nhà)

function User(fistname, lastname , avatar){

//tương tự Java

//Những cái chung}

* + - * 1. **Object Prototype**

-Nguyên liệu tạo nên ngôi nhà

-Cú pháp : Ten.prototype.tenKey = ‘value’

*-Không cần phải truyền vào.* Mặc định tất cả các đối tương được tạo từ OP sẽ mặc định có

|  |
| --- |
|  **Tiết kiệm bộ nhớ**: Các phương thức được định nghĩa trong prototype được chia sẻ giữa tất cả các đối tượng được tạo ra từ cùng một constructor, thay vì sao chép chúng vào mỗi đối tượng. |
|  **Tăng tính kế thừa**: Prototype cho phép các đối tượng kế thừa thuộc tính và phương thức từ các đối tượng khác, giúp dễ dàng mở rộng chức năng mà không cần thay đổi mã nguồn gốc. |
|  **Thêm phương thức động**: Bạn có thể thêm hoặc thay đổi các phương thức hoặc thuộc tính của tất cả các đối tượng thuộc cùng một constructor bằng cách thay đổi prototype của constructor đó. |

# **Date**

|  |  |
| --- | --- |
| Date() | Trả về kiểu String , thời gian hiện tại  var date = new Date() -> đối tương  var date = Date() -> kiểu chuỗi  date.getFullYear() : lấy năm  date.getMonth() : lấy tháng (nhớ +1)  date.getDate() : ngày |

Tài liệu : Javascript date object mozilla

# **Math**

|  |  |
| --- | --- |
| abs | Trị tuyệt đối |
| Ceil | Làm tròn trên |
| Floor | Làm tròn dưới |
| Random | Trả về 1 số thập phân ngẫu nhiên nhỏ hơn 1 |

# **Vòng lặp**

|  |  |
| --- | --- |
| for | Lặp với điều kiện đúng |
| For / in | Lặp qua key của đối tượng |
| For / of | Lặp qua value của đối tượng |
| While | Lặp khi điều kiện đúng |
| Do while | Lặp ít nhất 1 lần, sau đó lặp khi điều kiện đúng |

**For**

**For / in**

for(var key in object){} -> lấy ra các tên key

for(var key in array){} -> lấy ra các index của mảng

for(var key in string){} ->lấy ra các index của chuỗi

**For / of**

for(var key of array){} -> lấy ra các giá trị của mảng

for(var key of string){} ->lấy ra các giá trị của chuỗi

Object.key/values(ObjectName) : lấy giá trị chuyển thành mảng

**While**

**Do while**

# **HTML DOM (Document Object Model)**

Tài liệu : <https://www.w3schools.com/js/js_htmldom.asp>

When a web page is loaded, the browser creates a Document Object Model of the page.

DOM có 3 thành phần

* Element : có bn thẻ là bn element
* Attribute : thuộc tính trong các thẻ
* Text : Các chữ trong thẻ hoặc ngoài thẻ .Tóm lại là chữ



Mỗi ô hình vuông gọi là node

The DOM is a W3C (World Wide Web Consortium) standard. **Quy chuẩn của W3C**

 The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

**Lưu ý:** HTML DOM k phải của JS . JS chỉ cung cấp công cụ để truy xuất HTML DOM

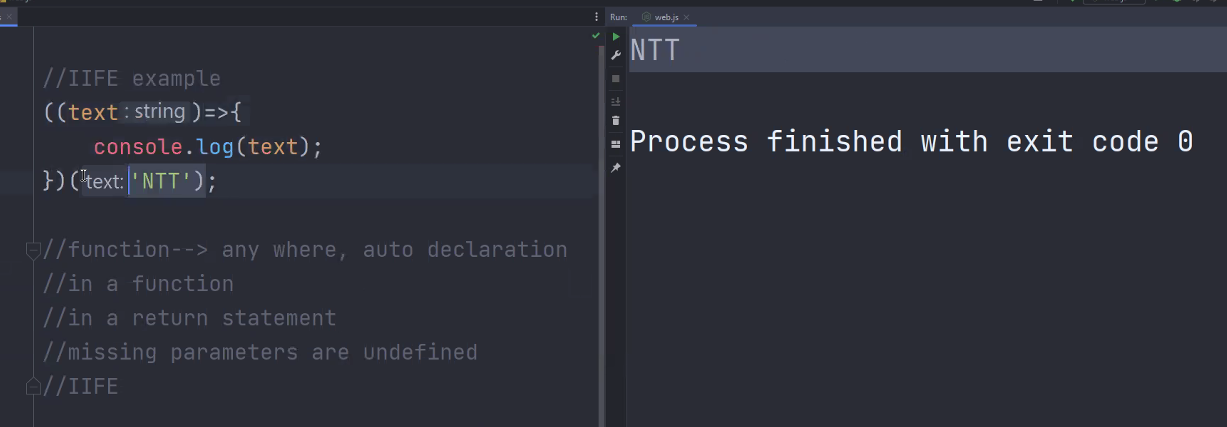
JS : Browser | Server (NodeJS) mà Server không có DOM -> JS k có DOM

**Element**

|  |  |
| --- | --- |
| querySelectorAll(‘duongDanThe’) | Lấy thẻ (tất cả). Trả về NodeList |
| querySelector(‘duongDanThe’) | Trả về 1 element duy nhất |
| getElementsByTagName(‘tenThe’) | Lấy thẻ (tất cả).Trả về HTML Collection |
| getElementById | Trả về 1 element duy nhất.Lấy thông qua id |
| getElementsByClassName | Lấy thông qua class. Trả về HTMLCollection |
| HTML collection (không quá quan trọng) | img, a, form  document.forms : in ra tất cả thẻ forms  Trả về HTML Collection |
| document.write(‘noiDung’) | Đoạn code JS ở đâu thì nó hiển thị ở đó.Nhưng thường đặt cuối nên nó hay ở cuối |

Nguyên tắc có cha lấy được con.

# **Tìm hiểu IIFE**

****