**Toán tử số học**

+ - \* /

%: chia lấy dư

++ -- :tang hoặc giảm 1 đơn vị

Toán tử ++ hoặc -- khi được dùng làm **tiền tố** (++a) sẽ thực hiện phép toán + hoặc - 1 đơn vị trước rồi mới trả về giá trị.

Còn khi được sử dụng làm **hậu tố** (a--) thì sẽ trả về giá trị trước rồi mới thực hiện + hoặc -.

**Toán tử gán**

= , += , -= , \*= , /= , \*\*=

**Toán tử nối chuỗi**

+ , +=

Toán tử so sánh:

==, !=, < , > , <= , >=,===,!==

==: so sánh trừu tượng: nghĩa là so sánh số 0 với ‘0’ trả về true, so sánh 1 chuỗi với 1 chuỗi khởi tạo = new String là true

===: so sánh nghiêm ngặt: trả về false

**Toán tử logical:**

&&, ||, !

|  |  |
| --- | --- |
| && | 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 |

**kiểu dữ liệu**

JavaScript có **8 kiểu dữ liệu cơ bản**, trong đó, có 7 kiểu dữ liệu nguyên thủy (**boolean, null, undefined, number, BigInt, string, symbol**) và 2 kiểu dữ liệu phức tạp (**object**). Và function

* Kiểu dữ liệu boolean: bao gồm hai giá trị true và false dùng để biểu diễn **đúng** và **sai**.
* Kiểu dữ liệu null: chỉ gồm một giá trị null, dùng để biểu thị trạng thái **không biết giá trị**.
* Kiểu dữ liệu undefined: chỉ gồm một giá trị undefined, dùng để biểu thị trạng thái **chưa gán giá trị**.
* Kiểu dữ liệu number: dùng để biểu diễn số nguyên hoặc số thực. Trong đó, số nguyên giới hạn trong đoạn ±(253-1.
* Kiểu dữ liệu BigInt: dùng để biểu diễn số nguyên lớn, có độ dài tùy ý. Để biểu diễn số thuộc kiểu BigInt, bạn chỉ cần thêm chữ cái n ở cuối.
* Kiểu dữ liệu string: dùng để biểu diễn kí tự, chuỗi ký tự,... bằng cách sử dụng dấu nháy đơn ('), nháy kép (") hoặc dấu "backtick" (`). Mỗi string có thể chứa một kí tự, nhiều kí tự hoặc không chứa kí tự nào (empty string).
* Kiểu dữ liệu symbol: dùng để tạo ra những giá trị duy nhất, thường dùng làm "key" cho object.
* Kiểu dữ liệu object: (array , object) :
* Kiểu dữ liệu function

Để kiểm tra kiểu dữ liệu của một biến, bạn có thể sử dụng typeof.

* Có hai dạng: dạng toán tử typeof x hoặc ở dạng hàm typeof(x).
* Kết quả trả về là một string ứng với tên của kiểu dữ liệu.
* typeof null trả về object. Đây là một lỗi sơ khai của JavaScript. Thực chất, null không phải là object.

String: toUpperCase(), length, trim()

Math: floor(), random(), ceil(): lam Nguyen,

Number: isInteger(),

*Su dung typeof a để kiểm tra kiểu dữ liệu của biến*

Giống nhau:

* Cả hai khi bị phủ nhận đều tra về true, nhưng không có cái nào bằng true hoặc false.
* Chúng đều đại diện cho một cái gì đó không tồn tại…

Khác nhau:

* null đại diện cho “nothing”, hoàn toàn không tồn tại, không xác định được thứ không được xác định.
* undefined thì có dạng data của riêng nó (undefined), null thì chỉ là một object
* null đưa về 0 khi vận hành bằng toán, undefined trả về NaN

**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.

1. false
2. 0 (số không)
3. '' or "" (chuỗi rỗng)
4. null
5. undefined
6. NaN

**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.

1. '0' (một chuỗi chứa số không)
2. ' ' (một chuỗi chứa dấu cách)
3. 'false' (một chuỗi chứa từ khóa false)
4. [] (một array trống)
5. {} (một object trống)
6. function(){} (một hàm “trống”)

**Biến**

Phạm vi biến:

**Toàn cục**: Biến toàn cục là loại biến mà nó được khai báo ở vị trí bên ngoài một hàm.

- Biến toàn cục có thể được truy cập ở bất kỳ vị trí nào trong chương trình JavaScript.

Khi ở bên trong một hàm, nếu ta khai báo biến mà thiếu từ khóa **var** *(hay nói cách khác là gán giá trị cho biến trong khi nó chưa được khai báo)* thì biến đó sẽ trở thành biến toàn cục.  
**cục bộ:**  Biến cục bộ là loại biến mà nó được khai báo ở vị trí bên trong một hàm.

- Biến cục bộ chỉ có thể được truy cập ở vị trí bên trong hàm mà nó được khai báo.

Vì biến cục bộ chỉ có thể sử dụng bên trong hàm mà nó được khai báo. Nên do đó, với cùng một tên biến, ta có thể khai báo ở những hàm khác nhau.

**Var:**

Phạm vi: cục bộ nếu trong hàm, toàn cục nếu ngoài hàm hoặc trong vòng lặp

Các biến có thể tái khai báo và cập nhật biến trong cùng phạm vi và sẽ không xuất hiện lỗi

Hành vi ngầm định của **Javascript** khi khai báo biên với **var** là chuyển vị trí khai báo biến hay khai báo hàm lên đầu của phạm vi trước khi thực hiện mã và trả về giá trị undefine

**Let:**

Phạm vi: Biến let nếu khai biến trong 1 khối mã thì chỉ có thể dung trong khối mã đó

**Let có thể được cập nhật nhưng không được khai báo lại**

 khi cố dùng biến let với trước khi khai báo biến sẽ nhận được lỗi tham chiếu

**Const**

Biến được khai báo với **const** chứa giá trị hằng (Giá trị không thể thay đổi trong khi thực hiện)

Giống với khai báo **let**, khai báo biến **const** chỉ được truy cập trong khối mã biến được khai báo

Giống với **let**, khai báo **const** được dịch chuyển lên đầu của đoạn mã nhưng không được khởi gán giá trị.

**Loop**

JavaScript hiện hỗ trợ 5 loại vòng lặp khác nhau:

+ **while**- vòng lặp while lặp lại một khối mã miễn là điều kiện được chỉ định đánh giá là đúng.

+**do...while** - vòng lặp do...while lặp khối mã một lần. Sau đó đánh giá điều kiện. Nếu điều kiện là đúng, lặp lại khối mã miễn là điều kiện là đúng.

+**for** - vòng lặp thông qua một khối mã cho đến khi bộ đếm đạt đến một số được chỉ định.

+ **for...in** - Vòng lặp for...in lặp qua thuộc tính của một đối tượng.

+ **for...of** - Vòng lặp for...of lặp các đối tượng có thể lặp như mảng, chuỗi, v.v.

**Vòng lặp:**

For…in: in ra index nếu mảng và string, in ra key nếu trong obj

For..of: in ra giá trị nếu là mảng và string, obj ko in ra đượcconst id = [[1,2],[3,4],[5,6]]

for(let [key,value] of id){

    console.log(`key: ${key}, value: ${value}`)

}

Trong vòng lặp ngta hay sử dụng 2 từ khóa là break và continue để thoát vòng lặp

Hoặc bỏ qua ở 1 đoạn nào đó

**Toán tử 3 ngôi:**

Biểu thức ? nếu đúng : nếu sai

**Hàm**

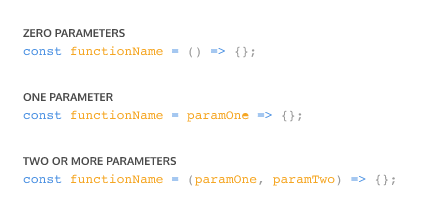
Hàm là 1 khối mã, làm 1 công việc cụ thể

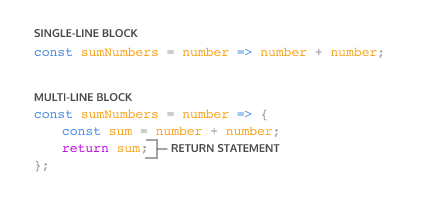
Loại hàm:

* Hàm có tham số : kq thay đổi theo tham số của hàm
* Hàm ko có tham số: kq của hàm ko bh thay đổi

Khai báo:

* Function delare: function name(para){} có thể dùng trước khi khai báo hàm
* Function expression: var a = function <name>(){} : tên có thể có hoặc k, phải khai báo bên trên r mới gọi hàm đc
* Arrow function:

return



Tham số: sd lúc định nghĩa hàm, ĐỐi số sd lúc gọi hàm

Đối tượng argument: lúc định nghĩa hàm ko truyền tham số nhưng lúc gọi hàm truyền tham số. nó là aargument(giống mảng) lúc xử lý dữ liệu thì có thể lấy ra từ argument giống như lấy ra từ mảng (for..of)

**String:**

Khởi tạo:

* Sdung toán tử gán: var a = ‘Hello’
* Sdung đối tượng String: var a = new String(‘Hello’) : kiểu dl là object

Backslash: Sử dụng \’ để dùng ‘, \\ để dùng \

Đo độ dài chuỗi: sd thuộc tinh .length của đối tượng String

TemplateString: ``

Method:

* .indexOf(‘’): xác định vị trí chuỗi
* -slice(index,indexb): cắt mảng từ a đến ptu b-1
* .replace(‘a’,’b’) : thay thế chuỗi a = chuỗi b
* toUpperCase()
* toLowerCase()
* trim() Loại bỏ khoảng trắng thừa 2 đầu
* split(‘’) cắt chuỗi thanh mảng và cắt tại điểm ‘’
* charAt() : xem kí tự tại index

**Number:**

|  |  |
| --- | --- |
| 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 |
| Number.prototype.toString() | Chuyển đổi và trả về số đã cho dưới dạng chuỗi |

**ARRAY:**

Kiểm tra mảng: Array.isArray(arr)

Độ dài mảng: .length

Lấy phần tử theo index: arr[2]

-toString (): chuyển thanh chuỗi

a.concat(b): nối mảng b vào sau mảng a

-join(‘’): chuyển thanh chuỗi và có dấu ngăn cách là ‘’

-push() : Thêm vào đuôi array

- pop(). Xóa phần tử cuối trả về pt đã xóa

-shift(): Xóa phần tử đầu tiên trả về pt đã xóa

-unshift()> them vào đầu array và trả về độ dài mảng mới

-slice(index,indexb): sao chép và cắt mảng từ a đến ptu b-1

Splice(index,a,’abc’):sẽ xóa đi a pt kể từ index, them vao vi tri day pt abc và trả về mảng mới

-indexOf():

-toSpring()

-map(): in ra 1 mảng mới với giá trị return

Filter(): in ra 1 mảng mới với giá trị thỏa mãn điều kiện ở return

forEach(): lặp qua các phần tử

findIndex(): tra ve index cua phan tu dau tien thoa man dieu kien

charAt(index): dung để kiểm tra phần thứ index của chuỗi  
reduce((bien luu tru, value),giatri ban dau cua bien luu tru): tra ve 1 ket qua duy nhat

some() kiểm tra xem có phần tử nào thỏa mãn đk ko(true/false)

every() kiểm tra xem mọi pt có thỏa mãn đk hay ko

find() Kiểm tra phần tu dau tien thoa man dieu kien va in ra phan tu day

**OBJECT**

Delete obj.thuoctinh

Delete obj[thuotinh]

Getter setter: khai báo hàm kèm theo từ khóa get ở trước. khi gọi thì gọi như thuộc tinh ko cần dấu ()

Set thì có tham số. khi gọi hàm thì set giá trị = dấu =

Factory Function:const FactoryF = (para1,para2) =>{ return{ key: para1, key: para2,f()}

Shorthand nếu key và value gống nhau thì return{para1,para2,f()}

Ktao: const a = FactoryF(…)

Constructor: function User(para1,para2){ this.para1 = para1,….}

Ktao: const a = new User(…)

Them thuộc tinh cho contructor. User.prototype.thuoctinh = value

Object.keys(obj) : in ra tất cả các key tạo thanh 1 mảng

Object.entries(obj): in ra mảng chứa nhiều mảng bao gồm cặp giá trị key value [[key,value][]]

Object.fromEntries(arr): ngược lại với obj.entries

Object.assign(obj1,obj2): hợp nhất 2 đối tượng, và đối tượng obj 1 sẽ bị thay đổi thanh giá trị mới còn obj2 thì k

**Callback**

Là 1 hàm, và nó được truyền qua đối số khi gọi 1 hàm khác

Callback có khá nhiều ứng dụng như sau:

* Gọi 1 hàm khi có 1 số sự kiện xảy ra. VD như khi click vào 1 nút ta gọi hàm A, khi tắt cửa sổ ta gọi hàm B, v…v.
* LINQ được xây dựng dựa trên khái niệm callback và lambda expression. Với callback, ta có thể thực hiện các thao tác như: Tìm 1 phần tử trong mảng, lọc phần tử trong mảng, sắp xếp mảng, … trở nên vô cùng đơn giản và thuận tiện.

**This:**

Trong **javascript**, chúng ta dùng từ khóa this để đại diện cho một đối tượng (Object). Nó sẽ trỏ về đối tượng mà nó đang thuộc về

Trong 1 phương thức this tham chiếu tới đối tượng sử dụng phương thức

Đứng ngoài phương thức this tham chiếu tới đối tượng window

khi ta khai báo biến global và hàm global, toàn bộ các biến và hàm đó sẽ nằm trong một object có tên là window nhưng trong strict mode thì nó ko phải là 1 đối tượng nên sẽ trả về undefine

This trong hàm tạo đại ddieenj cho đối tượng được tạo bởi hàm tạo đó

**So sánh 2 object**

* So sánh bằng tham chiếu: tham chiếu đến vùng nhớ
  + Toán tử bằng chính xác ===
  + Toán tử bằng ==
  + Hàm Object.is()
* So sánh thủ công: trỏ tới value của hai object để so sánh
* So sánh bằng Object.entries
* **So sánh bằng Object.keys**
* So sánh nông: dungf Object.keys để lưu các key vào 2 mảng tiến hành so sánh
* So sánh sâu: tương tự như ss nông nhưng ss sâu sẽ so sánh nhiều cấp của object(sd đệ quy)
* So sánh bằng JSON.stringtify console.log(JSON.stringify(fruit1) === JSON.stringify(fruit2));  
  //true. Nhưng nếu đảo vị trí thuộc tinh sẽ false

**Toán tử delete**

Trong mảng: Sử dụng **toán tử delete** không ảnh hưởng đến thuộc tính độ dài. Nó cũng **không ảnh hưởng** đến các chỉ số của các element tiếp theo. Vị trí bị xóa trở thanh pt undefine

Trong object: xóa cả giá trị của thuộc tính và chính thuộc tính đó.

**Closure:**

Theo khái niệm của nó thì chúng ta chỉ việc tạo một hàm nằm bên trong một hàm khác thì đó chính là closure. Closure có 3 scope chain, đó là: Có thể truy cập đến biến của chính nó (biến được định nghĩa trong dấu ngoặc nhọn của nó); Có thể truy cập biến của hàm bên ngoài; Có thể truy cập biến toàn cục (global).

#### Closures có thể truy cập biến của hàm bên ngoài ngay cả hàm bên ngoài đã trả về:

#### Closures lưu tham chiếu đến biến của hàm bên ngoài

**Async:**

Sync: đồng bộ nghĩa là đoạn mã nào viết trước chạy trước, đoạn mã nào viết sau chạy sau

Async:  tức là code chương trình không hẳn tuần tự nữa, nhiều lệnh có thể thực hiện cùng lúc. Có khi lệnh dưới cho kết thúc và cho kết quả trước cả lệnh phía trên.

Xử lý bất đồng bộ:

* Dùng hàm Callback
* Sử dụng Promise
* Dùng async/await (của es6)

**HTML DOM**

**DOM** là tên gọi viết tắt của (**D**ocument **O**bject **M**odel – tạm dịch Mô hình Các Đối tượng Tài liệu). Là một chuẩn được định nghĩa bởi W3C (Tổ Chức Web Toàn Cầu – World Wide Web Consortium). DOM được dùng để truy xuất và thao tác trên các tài liệu có cấu trúc dạng HTML hay XML bằng các ngôn ngữ lập trình thông dụng như Javascript, PHP…

Gồm 3 thành phần: element, attribute, text (Node)