Javascript

# Lý thuyết

* Syntax Parser: Là 1 chương trình có thể đọc và hiểu được code mình viết ra nếu viết đúng cú pháp.
* Lexical Environment: Là nơi lưu trữ, xác định vị trí biến, function nằm ở đâu trong 1 file code.
* Excution Context: là việc thực thi lệnh như thế nào, chẳng hạn tuần tự như thế nào. Biến nào trước, biến nào sau. Code JS chạy được là nhờ “Excution Context”.
* Name-Value Pair: cặp tên – giá trị: ví dụ address = ‘Ho Chi Minh’. Thì address: name, ‘Ho Chi Minh’: value. Khi khởi tạo mà không gán thì có giá trị là undefined
* Object: là 1 tập hợp các Name-Value Pair.
* Global Object: Syntax Parser sẽ chuyển code sang mã máy, do không có code nên nó bỏ qua và không có lỗi. Tiếp theo “Global Execution Context” sẽ tạo cho bạn 2 thứ là Global Object là window và biến là this có giá trị bằng với window luôn. Global Object là do máy tự tạo ra cho mình. Khi chạy JS code trên trình duyệt nên Global Object là window (Lưu ý: chỉ có JS mới như vậy). Global là toàn cục, những gì không nằm trong function
* Hoisting: hoisting là cơ chế của JavaScript cho phép các khai báo biến hoặc hàm được dời lên trên đầu phạm vi của chúng trước khi thực thi đoạn code. Lưu ý: Là cơ chế này chỉ di chuyển phần khai báo mà thôi còn các phần khác giữ nguyên không đụng gì đến nó hết.

Ta có thể sử dụng biến trước khi khai báo nó:

*thuongHieu = "NIIT";*

*console.log(thuongHieu);*

*var thuongHieu;*

Hành vi Hoisting đối với hàm:

* Khai báo hàm:

function abc(){} – tương tự với khai báo biến, có thể đặt trước hoặc sau lệnh sử dụng

* Khởi tạo biểu thức hàm:

var abc = function(){} Phải đặt trước lệnh sử dụng.

var abc = function abc(){}: cũng không được sử dụng

Độ ưu tiên của hosting:

Phép gán ưu tiên hơn khai báo hàm

Biểu thức hàm ưu tiên hơn gán biến

Khai báo hàm ưu tiên hơn khai báo biến

* Biểu thức hàm > Gán biến > Khai báo hàm > Khai báo biến

Biểu thức hàm:

var message = function() {

document.write("Đây là hàm");

}

Gán biến:

message = "Đây là biến chuỗi";

Khai báo hàm:

function message() {

document.write("Đây là hàm");

}

Khai báo biến

var message = "Đây là biến";

Cách khắc phục:

Sử dụng “use strict”. Hoặc sử dụng “let”, “const” cho bất kỳ khai báo biến nào. Let và const cũng được đưa lên đầu nhưng chỉ được dùng khi đã khai báo

* Execution stack:

Mỗi một hàm thực thi ( invoke function ) sẽ có một Execution Context được tạo ra và xếp vào một stack. Khi hàm chạy xong, Execution Context chứa hàm đó sẽ bị pop ra ngoài.

Tức là sẽ có 1 stack chứa các invoke function theo tuần tự được gọi ra trong code file. Khi 1 hàm execution thì nó sẽ bị đẩy ra ngoài stack

* Scope Chain

trong một Execution Context, nếu ta truy cập giá trị một biến, mà không tìm thấy biến đó trong Execution Context hiện tại thì nó sẽ tìm ở Outer Environment (giống như Lexical Enviroment bao gồm “Global Object” và biến “this”).

Bài tập: <https://github.com/Bloc/mentor-exercises/blob/master/exercises/javascript/closure-scoping-exercises.md>

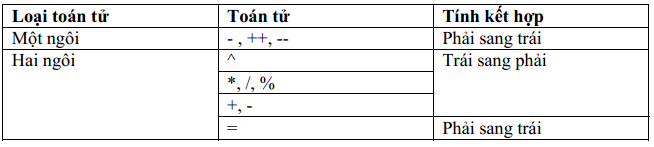
* dynamic typing: nghĩa là kiểu dữ liệu không được định sẵn trước khi gán. Biến chỉ là cái nhãn, kiểu dữ liệu sẽ được định nghĩa khi runtime.
* Primitives Type: undefined, null, Boolean, string, number, sysbol(ES6)

Null: typeof null là object, chưa biết kiểu dữ liệu gì thì gán = null

Undefined: typeof là undefined

* Toán tử và thứ tự ưu tiên:
* Toán tử gán: ‘=’ . Nâng cao: ++, --, +=, -=, \*=, /=
* Toán tử toán học: + - \* /
* Toán tử quan hệ: >, <, >=, <=, ==, !=, ===
* Toán tử luận lý: and(&&), or(||), not (!(a>b))

Độ ưu tiên: Toán tử 1 ngôi ưu tiên hơn toán tử 2 ngôi



* Coercion: ParseInt(), Number(), Stringify()

# Object

1. Tạo đối tượng:

Có 3 cách tạo đối tượng:

* const dog = {}
* const cat = new Object();
* const horse = Object.create({ })

1. Truy cập thành phần của Object

Có 2 cách truy cập 1 Object:

* objectName.propertyName
* objectName["propertyName"]

1. First class function

Functions (hàm) trong Javascript có thể được lưu trữ như biến, truyền như là tham số vào hàm khác, là kết quả trả về của một hàm, lưu trữ dữ liệu hay thậm chí là có thuộc tính riêng như đối tượng (objects).

Detail and example: [Click here](Ref-detail/First-class%20function.docx)

* Function Declaration và Function Expression:
* Function declaration: function đó có tên
* function handleClick(){}
* Function expression
* const handleClick = function(){console.log(“click”) }
* function anonymous: function loại này được sử dụng và biến mất ngay sau đó

const handleClick = () => {

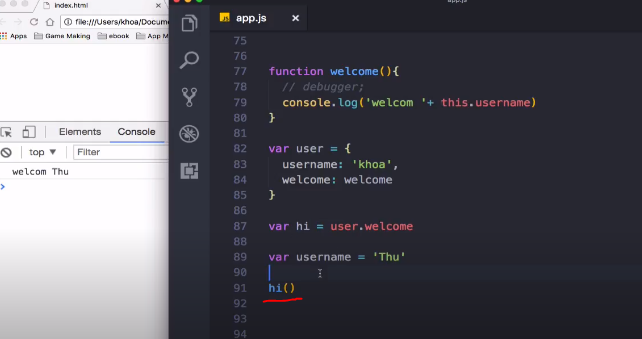
console.log('clicked')

}

* Call-site là gì: là nơi hàm được gọi
* Implicit Binding: Xác định “this” đang tham chiếu đến cái gì bằng cách xem đối tượng bên trái của hàm sử dụng dấu chấm

Detail: [Click here](Ref-detail/Implicit-Biding.docx)

* Implicit lost là gì: Hãy xác định hàm gọi ở đâu? Hàm hi() này đang gọi ở global nên kq là username = “Thu”



# This trong javascript:

“this” cho phép:

* sử dụng lại function trong các ngữ cảnh khác nhau
* xác định tập trung vào method nào khi gọi method

Example:



Khi user.showName(); được gọi thì “this” ở đây trỏ đến “firstname” và “lastname” trong “user” object.

Khi “showNameInVietnamese()” được gọi thì “this” ở đây trỏ đến “firstname” và “lastname” trong global object.

# Hiểu thêm về hàm

1. Function Arguments: có thể truyển bao nhiêu đối số cũng được, kể cả khác kiểu dữ liệu

Example: [Click here](Ref-detail/Example-function-arguments.docx)

1. Function overloading: Khi có 2 hàm trùng tên thì nó sẽ lấy cái dưới cùng do chạy từ trên xuống dưới
2. ES6 default arguments: Ta gán giá trị mặc định vào parameter luôn.

Example: function pay(total, tip = 0)

1. Immediately invoked function expression: là hàm được chạy ngay khi nó dược định nghĩa

Example: [Click here](Ref-detail/Immediately-invoke-function-expression.docx)

1. Closure: Hàm lồng hàm khép kín. Return về 1 hàm

Example: [Click here](Ref-detail/Closure.docx)

1. Closure – return object: Return về 1 object có hàm

Example: [Click here](Ref-detail/Closure-return-object.docx)

1. Closure – return mảng function: Return về 1 mảng hàm. Example: [Click here](Ref-detail/Closure%20-%20return%20mảng%20function.docx)
2. Callback: Callback là một hàm sẽ được thực hiện sau khi một hàm khác đã thực hiện xong - vì thế nó có tên là callback.
3. Bind, call, apply: [Click here](Ref-detail/bind,%20call,%20aplly.docx)

Bind: gắn kết các hàm hoặc object lại với nhau

1. Function borrowing là gì: [Click here](Ref-detail/function%20borrowing.docx)

# Hướng đối tượng trong Javascript

1. Tổng quát về hướng đối tượng

Quy tắc:

- Trừu tượng (abstraction)

- Đóng gói (encápulation)

- Kế thừa ( inheritance )

- Đa hình (polymorphism)

1. Prototype Inheritance là gì? Prototype chain là sao?: [Click here](Ref-detail/Prototype%20Inheritance.docx)

Khi một Object, array, string, number, boolean được tạo ra thì sẽ có 1 thuộc tính là object proto{}. Nó bao gồm chứa các hàm như ở Object: “bind”, “call”, “apply”…. Array: “push”, “concat”…

1. Reflect: [Click here](Ref-detail/Reflect.docx)

Reflection in Javascript -> View/Edit (View: property và method): Có thể xem và chỉnh sửa

1. Function constructor và từ khóa new: [Click here](Ref-detail/Function%20constructor%20và%20từ%20khóa%20new.docx)
2. Sử dụng Prototype như thế nào: [Click here](Ref-detail/Sử%20dụng%20prototype%20như%20thế%20nào.docx)
3. Tạo primitive type từ function constructor?: Click here

# DateTime

* 4 cách tạo 1 object Dateime:

new Date()

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

new Date(milliseconds)

new Date(date string)

* UTC

new Date("2014-01-01") => 31/12/2013 theo giờ UTC

Hàm

.getDay(): Thứ

# Function

Function là Object

Functions (hàm) trong Javascript có thể được lưu trữ như biến, truyền như là tham số vào hàm khác, là kết quả trả về của một hàm, lưu trữ dữ liệu hay thậm chí là có thuộc tính riêng như đối tượng (objects).

* Lấy element thông qua selector
* Id: document.getElementById(id)

* Tâp hợp tất cả các element con trong 1 element cha
* element.childNodes[0]
* Lấy data, text trong 1 element
* .data

Muốn set lại data của textNode thì dùng

const textNode = element.childNodes[0]

textNode.data = text;

* String
* text[5]: lấy ký tự thứ 5 của text
* text.length: độ dài của text
* substring(): Lấy ra một chuỗi con từ 1 chuỗi cha
  + str.substring(1, 3): Lấy ra một chuỗi từ index 1 đến 2
  + str.substring(2): Lấy ra 1 chuỗi từ index 2 trở đi
* .split('.'): cắt chuỗi tại dấu “.”
* .pop(): lấy phần tử cuối cùng trong array
* String(): convert -> string
* .toLowerCase(): in thường
* .toUpperCase(): in hoa
* .repeat(n): lặp lại chuỗi str n lần - str.repeat(n)

Unicode:

* .charCodeAt(): lấy mã unicode của một chữ
* String.fromCharCode(char): chuyển số thành chữ
* Array
* [array[1], array[2], array[0]]: khởi tạo
* arr.pop(): Lấy phần tử cuối cùng trong mảng
* reverse(): đảo ngược 1 chuỗi
* sort(): sắp xếp

list\_code.sort(function(a, b){ return a - b}) : Sắp xếp mảng tăng dần

* .indexOf(1) : nums.indexOf(1) – tìm index của số “1” trong mảng nums
* .map(): return mảng mới với điều kiện đã cho - const map1 = array1.map(x => x \* 2);

str\_ara.map(v => max = Math.max(max, v.length));

* [...to\_string]: clone 1 mảng
* .splice(2, 1): xóa 1 phần tử ở vị trí index = 2
* URL
* document.URL: lấy URL.
* document.URL.href: lấy href current

Tip:

* Xóa element empty trong array

var split\_href = href.split('/').filter(item => item);

# Statement

* For

for ( var i=1; i<=10; i++ ){

list\_btn.push(`<button>${i}</button>`)

}

* For in

var list\_btn = [‘1’,’2’, ‘3’]

for ( element in list\_btn ){

console.log(list\_btn[element])

}

“element” là key, “list\_btn[element]” là value

* throw

# Method

* setInterval(): thực hiện 1 hành động gì đấy được lặp lại trong 1 khoảng thời gian nhất định.
* setTimeout: chờ một khoảng thời gian thì thực hiện
* clearTimeout: hủy bỏ lệnh setTimeout
* typeof variable\_name: kiểm tra kiểu dữ liệu của biến

# Math

* Math.random() \* 20: random number từ 0->19. Math.floor(Math.random() \* 20) + 1
* Math.floor: làm tròn số (5.95->5; 5.05->5; 5->5; -5.05->-6
* Math.abs(): trả về giá trị tuyệt đối của 1 số
* Math.max(): tìm giá trị lớn nhất trong mảng - Math.max(max, temp) .

# ES6

* (/a...b/).test(str) : Kiểm tra chuỗi “str” có ký tự “a” và “b” phân cách nhau bởi 3 ký tự ở giữa hay không
* Array
* .includes(1): nums.includes(1) – kiểm tra trong mảng có số “1” không. !nums.includes(1): phủ định

# Regular Expressions

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

<https://regexr.com/>

* Khởi tạo:
* let re = /ab+c/;
* Cú pháp:
* /[^p]/g

Example:

- Kiểm tra trong câu ký tự đầu tiên có viết hoa không: /^([A-Z])/

- Kiểm tra trong câu ký tự cuối cùng có phải là dấu chấm không: /[.]$/

# Other

* NaN: Khi check == NaN thì dùng !NaN
* Lấy style của một element:

var element = document.querySelector(".line-shadow-white").style.background = 'url("file:///E:/Growme-core/SourceCode/static/js\_create\_thumb\_templates/img/slide.png") repeat scroll 0% 0% / cover padding-box border-box, url("file:///E:/Growme-core/SourceCode/static/js\_create\_thumb\_templates/img/slide-line.png") repeat scroll 0% 0% / cover padding-box border-box, rgba(192, 192, 192, 0.08) none repeat scroll 0% 0% / cover padding-box border-box';

* Tạo một dãy button bằng js

var element\_list\_btn = document.getElementById("list-btn")

var element\_node = element\_list\_btn.childNodes[0]

var list\_btn = []

for ( var i=1; i<=10; i++ ){

list\_btn.push(`<button>${i}</button>`)

}

for ( element in list\_btn ){

element\_list\_btn.innerHTML += list\_btn[element]

}

* A variable using a user-defined name

var var\_name = 'abcd'

var n = 120;

this[var\_name] = n;

console.log(this[var\_name]);

* Toán tử bậc 3:

nums[0] > nums[2] ? nums[0] : nums[2]

* Swap – giữ được vị trí các phần tử khác trong arr

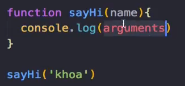
function swap(arr){

[arr[0], arr[arr.length - 1]] = [arr[arr.length - 1], arr[0]];

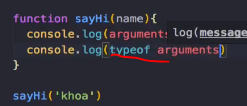
return arr

}

* debugger: Bật debugger lên để debug
* arguments: in ra đối số



* typeof : in ra kiểu dữ liệu



* Biến vẫn có thể tùy

let middle = arguments[2] || ‘ ‘

middle có thể là argument[2] hoặc là ‘ ’

* Trong JS ko có sleep nên tự custom

Ảnh có chứa văn bản

Mô tả được tạo tự động