Cours développement front avancé

ECMAScript = standard de l'organisation ECMA, forme la base de JavaScript.
Inventé par Brendan Eich

Les plateformes capables de lire de l'ECMAScript sont **nodeJS** et **les navigateurs**

Les + de l'ECMAScript :

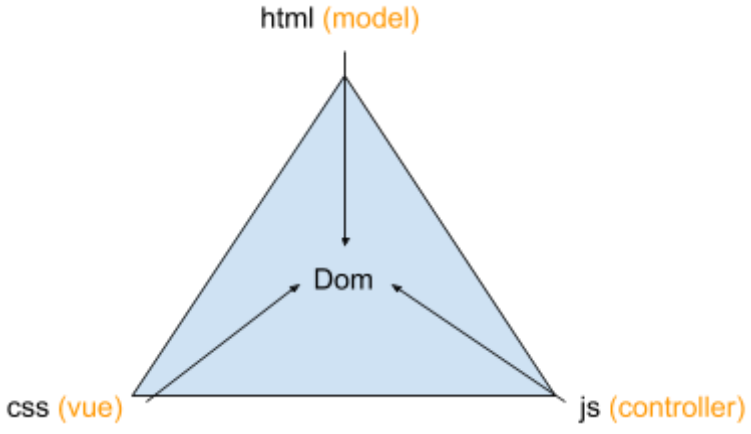
- **Fullstack**
- **L'aspect asynchrone : non bloquant (ex : le cloud) -> AVENIR , rend le js très important**

Aujourd'hui -> Cloud + scability = virtualisation (vps)
Demain -> Le monde ?

flexbox : http://flexboxfroggy.com/

Contexte d'évaluation de l'ES :

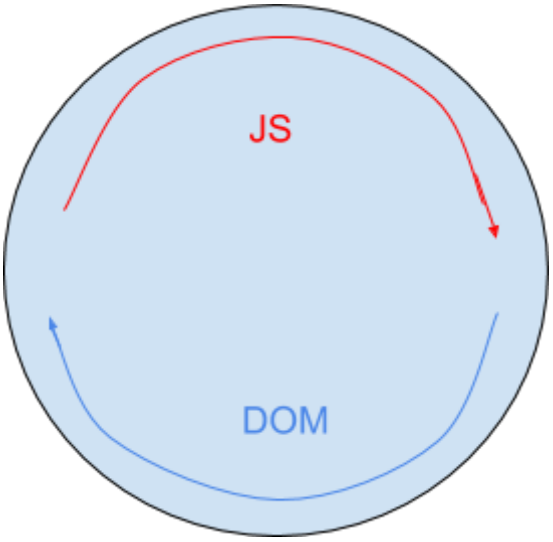
HTML (model -> référencement naturel)**, CSS** (vue)**, JS** (controller) **-> DOM** (Document Object Model) (lien entre les éléments, récupère les éléments et les mets en pages -> tout le temps présent et tout le temps modifiable)



Le seul langage pouvant être interprété par la machine est le binaire (code machine)
Js : utilise une virtual machine

Cycle Mono Thread de la virtual machine: le rafraîchissement permet d'exécuter des choses en continu -> cycle ininterrompu
Attention, toutes les boucles sont bloquantes pour le thread (while,for)

Elastic racetrack (schéma ci-dessous)



Pour connaître le taux de rafraichissement d'un navigateur:

```
setInterval(function(){console.log(3);}, 0)
//faire un console.log de "3" toutes les 0 millisecondes (donc le plus vite possible)
```

on utilise la fonction setInterval: et on force l'interval à 0

```
var start = new Date();
setInterval
(
function()
        {
        var end = new Date();
        console.log(end - start);
        start = end;
        }, 0
)
```

- asynchrone : reporter d'un cycle
- function : permet d'être rappelé

Callstack = la pile d'appels : c'est l'ordre dans lequel les opérations sont exécutées.
https://blog.sessionstack.com/how-does-javascript-actually-work-part-1-b0bacc073cf

Note : placer les script tout en bas de notre <body> :

- le DOM se charge même si le script est buggé
- cela permet d'afficher un élément du DOM avant de charger les script (avantageux pour le temps de chargement en mobile VS la patience des internautes)

attribut async = télécharger le script en parallèle de l'affichage du DOM
defer : attend que le dom soit rendu avant d'exécuter le script
<script async defer></script> Permet d'être placé dans la balise <head>
http://www.growingwiththeweb.com/2014/02/async-vs-defer-attributes.html

Valeur d'une variable (var) par défaut = undefined

var i = 1;

'=' est un opérateur d'affectation

En Ecmascript il y a une affectation **par valeur** et une affectation **par référence**

- valeurs primitives affectées par valeur : chaînes de caractère, nombres, boolean, undefined et null
- valeurs complexes affectées par référence : array, object, function
  https://www.youtube.com/watch?v=_P7S2lKif-A (exemple de référence qui me manquait !)
  Si ça peut vous permettre de réviser vos références :
  https://www.youtube.com/watch?v=wO89_H7GqaQ
  https://www.youtube.com/watch?v=AIXUgtNC4Kc

Déclaration de fonction nommée :

```
function hetic () {
        return "OK";
}
hetic();
```

Expression de fonction :

```
var hetic2 = function() {

};
```

-> undefined

IL FAUT ÉVITER DE FAIRE DES VARIABLES GLOBALES

LES VARIABLES GLOBALES, C'EST MAL !

```
(function() {
	"use strict";
	// IIFE
	// Immediately Invoked Function Expression
	var i = 1;
	setInterval (
		function() {
			console.log( i++ );
		}, 500
	)
} ) ();
```

La variable i est reconnue grâce à **la closure** : étant donné qu'elle se situe dans la même fonction parente que la fonction qui appelle cette même variable.

Lire : https://github.com/getify/You-Dont-Know-JS,
http://speakingjs.com/

namespace :
Un namespace peut être déclaré de cette façon : var hetic = hetic || {};

fichier index.js :

```
var hetic = hetic || {};

(function() {
	"use strict";
	// IIFE
	// Immediately Invoked Function Expression
	var i = 1;
	var init = function() {
		move();
	}

	hetic.move = function() {
		return { x : i++, y : 100 }
	}
} ) ();
```

fichier start.js :

```
var hetic = hetic || {};

(function() {
```

```
        setInterval (
                function() {
                        console.log( hetic.move() );
                }, 500
        )

})()
```

02/11/17

Listes d'action pour créer un projet avec npm:

Créer un dossier :

step 1 : mkdir [nom du dossier] *//créer un nouveau dossier*
step 2 : cd [nom du dossier]/
step 3 : npm init
step 4 : description, keywords → référencement
step 5 : license → MIT *//c'est un type de licence parmi d'autres*

→ génère un .json "*package.json*" dans le dossier

Principe de la numérotation des versions (SemVer) :

Chiffre 1 : version majeur
Chiffre 2 : version de bord
Chiffre 3 : correction de bug
Exemple : 1.0.0

Le fichier *package.json* est considéré comme le point central de la programmation ES. Il contient toute la config du projet.

Browserify

Browserify : http://browserify.org/
Ligne de commande : *npm install -g browserify* *//installe globalement '-g'* global à l'ordinateur
Ligne de commande : *npm install --save browserify* *//écrit dans le json, nécessaire pour que les dépendances fonctionnent '--save'. Les personnes utilisant le projet seront qu'on a utilisé ce package pour faire fonctionner le projet.*
Ligne de commande : *npm install uniq --save*
Ligne de commande : *node main.js* *//on lance le programme avec node*
Ligne de commande : browserify main.js -o bundle.js

Le bonus Patrick :
sudo = “je suis super administrateur” :’)
A éviter au maximum, car ouvre des droits trop importants. Il faut s’en passer si possible lors de l’emploi d’utilitaires en ligne de commande.

Arbre de dépendance = arborescence

Browserify donne accès à une nouvelle commande “require()” qui n’est pas native à JS mais qui le devient.

require : prend une librairie existante et l’importe dans le fichier de manière à donner accès à ses fonctionnalités

Compilation du code :

```
browserify  main.js  -o  bundle.js
```

permet de compiler le code JS afin d’avoir un code reconnu par le navigateur.
Dans l’exemple ci-dessus : main.js étant le fichier à compiler et bundle.js le fichier dans lequel on compile. J’ai faim.

PHP -S 0.0.0.0:1234 //lancer un serveur sur l’ip *0.0.0.0* et sur les ports *1234*

Comment utiliser uglifyify ?
C’est à dire ajouter une phase de minification à browserify.
Ligne de commande :
sudo npm install -g uglifyify, puis npm install --save uglifyify
browserify -g uglifyify ./main.js > bundle.js
La “>” étant une redirection de main.js vers bundle.js

Ajouter cette ligne dans le package.json :

```
"scripts": {
  "start": "browserify -g uglifyify ./main.js > bundle.js",
```

La clé “start” permet d'effectuer une liste de commande de cette manière :
npm start sans avoir besoin de réécrire toute la ligne

modul.exports = {
divide : divide,
multiply : multiply
}

sur une autre page :

var math = require(./math); (require le nom du fichier)

03/11/17

ORIENTATION OBJET

Une classe = un moule, une template.
C'est une structure qui est instanciable pour former un objet.

var redCar = new Voiture();

1- **Héritage** : Les enfants hérite des propriétés du parent.
2- **Encapsulation** : permet de protéger les valeurs : setter/getter méthode accesseurs.
exemple : *setRoue(n);*

Mauvaise manière :

```
var maternelle = {
        eleves : [ ]
        ,addEleve : function (eleve) {
                if (typeof eleve === 'string') {
                        this.eleves.push(eleve)
                }
        }
        ,getEleves : function() {
                return this.eleves;
        }
}
```

Bonne manière pour éviter de briser l'encapsulation :

```
var maternelle = {
        eleves : [ ]
        ,addEleve : function (eleve) {
                if (typeof eleve === 'string') {
                        this.eleves.push(eleve)
                }
        }
        ,getEleves : function() {
                // Casser la référence
                return this.eleves.concat();
        }
}
```

3- **Polymorphisme** :
Les classes sont nommées par convention avec des capitales en début de mot, exemple :

Toutes les propriétés sont déclarées dans la fonction constructrice.
Propriété :

```
var Perso = function( nom ) {
        this.name = nom;
}
var gerald = new Perso ('nom');  //gerald.name ( retourne 'nom')
```

```
Perso.prototype // renvoie un objet vide avec un construtor
```

Toutes les méthodes doivent être déclarée en dehors de la classe, au sein de l'object prototype.
Méthode :

```
Perso.prototype.bouge = function() {
        return this.name + " bouge";
} //gerald.bouge ( retourne 'nom bouge')
```

Autre méthode :

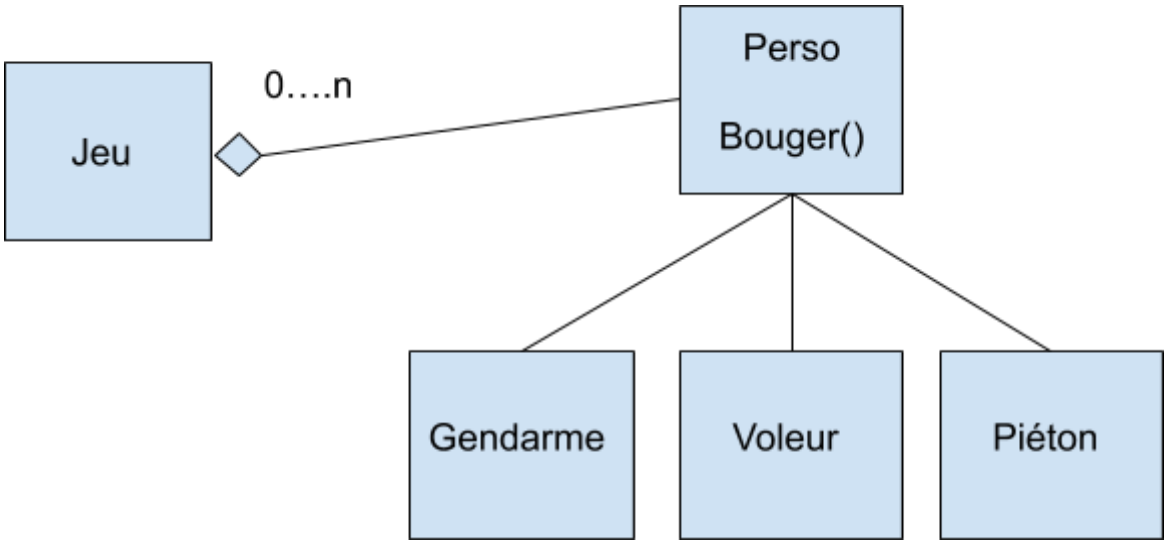
```
Perso.prototype = {
        bouge : function() {}
        , respire : function() {}
}
```

Livre Orienté objet : Tête la première design patterns
tête la première - design patterns - ebook

4- **Composition**

EXEMPLE DE JEU REPOSANT SUR LE POLYMORPHISME :

UML



Jeu Gendarme & Voleur :

```
var Perso = function ( name ) {
        this.name = name;
        this.x = 0;
        this.y : 0;
};
Perso.prototype =  {
        move : function(x,y) {
                this.x += x;
                this.y += y;
                return this.name +'bouge.'
        }
        ,setName : function (name) {
                if ( typeof name == 'string' ) {
                        this.name = name;
                }
        }
};
var Gendarme = function (name) {
        // Méthode simple
        this.name = name;

        // Méthode optimisée
        Perso.apply( this, [ name ] );
};
```

```
// ATROCE
// Gendarme.prototype = new Perso()
Gendarme.prototype = Object.create( Perso.prototype);   //Gendarme devient Perso pas
besoin de dupliquer le code du coup
//override
Gendarme.prototype.bouge = function (x,y) {
        this.x += x;
        this.y += y;
        return this.name +'bouge un peu aviné.'

}
var Voleur = function (name) {
        this.name = name;
}
Voleur.prototype = Object.create( Perso.prototype);
//override
Voleur.prototype.bouge = function (x,y) {
        this.x += x;
        this.y += y;
        return this.name +'bouge furtivement.'

}
var Pieton = function (name) {
        this.name = name;
}
Pieton.prototype = Object.create( Perso.prototype);

var Jeu = function () {
        this.persos = [ ];
        this.addPerso (new Gendarme('robert'));
}
Jeu.prototype.addPerso = function( p )  {
        if (p instanceof Perso) {
                this.persos.push( p )
        }
}
Jeu.prototype.start = function () {
        var self = this;
        this.interval_ID = setInterval {
                function() {
                        var p;
                        for (var i = 0, l = self.persos.length; i < l; i++) {
                                p = self.persos[ i ];  //on met this pour faire référence à la
                                                        function
                                p.move() //POLYMORPHISME
                        }
```

```
                }, 500
        }
}
Jeu.prototype.stop = function() {
        clearInterval (this.interval_ID);
}
```

pour commencer le jeu : *var jeudelamortquitue = new Jeu();*
jeudelamortquitue.addPerso(new Voleur('banchaa'));
jeudelamortquitue.addPerso(new Pieton('mona'));
jeudelamortquitue.addPerso(new Gendarme('abraham'));
jeudelamortquitue.start();
jeudelamortquitue.start();

ordonnée : array , on les appelle avec 1,2,3 ...
désordonnée : objet on les appelle par leur nom
composition

faire le jeu du snake : un fifo

20/11/17

METEOR

VOCABULAIRE

Gestion de persistance (côté serveur) : base de donnée

Logique applicative : logique de l'application

Routage : roots (URL qui capte les messages entre serveur et client)

DOM : gestion de l'affichage avec javascript pour afficher des données en temps réel sur le client

Application isomorphe (Meteor) : La gestion de programmes est disponible côté client et serveur

Web socket : protocole temps réel au sein du HTTP (permet d'échanger des données entre client et serveur directement) appelé "push" (le serveur envoie une information au client

sans que celui-ci ait à lui demander cette information. (ex : slack, dès qu'une modif est faite tous les utilisateurs le savent)

Full Stack Reactivity : chaque fois qu'une donné change, ca change l'interface sans changé le code

INSTALLER METEOR

http://ericpriou.net/formations/meteor/fr/intro/install.html
puis : *create meteo isoapp*
puis : *meteor*
puis : meteor build --directory ../
si on veut installer npm : meteor npm install lodash --save

Pour Windows :

step 1 install choco :

@"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile -InputFormat None -ExecutionPolicy Bypass -Command "iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))" && SET "PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin"

step 2 install meteor :

choco install meteor

step 3 crée un application :

meteor create [nom de l'app]

Pour OS X :

étape 1 :

https://www.meteor.com/install
ouvrir le shell et rentrer la ligne suivante :

```
➜ ~ curl https://install.meteor.com/ | sh
```

Une fois terminé, meteor est installé sur votre machine

étape 2 :

pour créer un nouveau projet, entrer la ligne suivante :

```
➜ Projets meteor create nom_application
```

faites ensuite *"**cd nom_application**"* afin de vous retrouver à la racine du projet créé

étape 3 :

pour lancer le serveur local et ainsi pouvoir développer votre application avec un retour navigateur en auto-refresh, tapez:

```
➜ nom_application meteor
```

le serveur local sera de base lancé sur le port 3000, mais changera si déjà utilisé, référez vous à votre terminal

étape 4 :

pour déployer l'application, il faut d'abord minifier le code et préparer un dossier particulier au déploiement. Pour ce faire:

```
➜ nom_application meteor build --directory ../
```

meteor va créer un dossier *"**bundle**"* à la racine précédant votre dossier d'application (ici dans Projets/)
Il ne vous reste plus qu'à importer votre dossier *"**bundle**"* sur un logiciel FTP.

MongoDB

1- Lancer le serveur meteor
2- Fichier main.js à la racine du dossier :

```
var students = new Mongo.Collection( "students" );
students.insert( { name: "eric" } );
```

3- Autre terminal : cd nom_application
meteor mongo
4- Dans mongoDB : db.students.find().pretty() // ressort le(s) objet(s) au format JSON dans le shell

https://www.youtube.com/watch?v=dQw4w9WgXcQ