

Javascript

Introduction

Historique

- Langage créé en 1995
 - et en quelques mois
 - Par Brendan Eich
- Pas de lien direct avec Java
- Faiblement typé
- Objets dynamiques
- Les fonctions sont des variables
- Utilise le **Prototypage**

Les outils de développement

Chrome Dev Tools

- Firebug : 2006
- Recopié par Google (Dev Tools)
- Capture des Elements
- Ecoute du Network
- Breakpoints sur les sources
- Utilisation de la console
 - `console.log()`, `debugger`, `console.table()`
- Profiler

Les IDE

- Netbeans
- WebStorm / IntelliJ

Et si on veut des problèmes :

- Eclipse
- Sublime Text
- Vim, Emacs, Notepad++ ...

Les bases du langages

Premieres lignes

- Utilisation d'un navigateur
- Syntaxe *à première vue* proche de C
- Utilisation des accolades
- Utilisation de `for` ou `while` classique

```
var x=0;
var text="bonjour";
console.log(text+x);
for (var i=0;i < 5 ; i++){
    x += i;
}
```

Différences notables avec C

- Déclaration des variables non typées
- Typages faibles
- Déclaration facultative
- Les fonctions sont des variables
- ; conseillé mais facultatif

```
var add = function(x,y){  
    return x+y;  
}  
var result = add(2,4);  
console.log(result);
```

Problèmes récurrents

- Langage créé en quelques mois (1995)
- Pas de lien direct avec Java
- Problèmes d'interpretation
- Langage verbeux
- TRES difficile à debugguer
 - Bien structurer le code
- Tres facile à tordre
 - Bien... ou pas

Intégration avec HTML

Client ou serveur ?

- Javascript est designé pour HTML
- Il peut néanmoins être utilisé côté serveur
- ...Avec des soucis de multithreading

Code à l'intérieur de la balise

```
<html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <link rel="stylesheet" href="css/styles.css" type="text/css" />
  <script type="text/javascript">
    console.log("Weather is not marvelous at all today");
    alert("Hey, what did you expect ?");
  </script>
</head>
<body>
  <h1>Hello World</h1>
</body>
</html>
```

- Le code est exécuté dès l'apparition de la balise
- Plusieurs balises peuvent se succéder
- Elle ne s'executent pas en parrallèle

La balise script

```
<html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <link rel="stylesheet" href="css/styles.css" type="text/css" />
  <script type="text/javascript" src="js/prettify.js"></script>
</head>
<body>
  <h1>Hello World</h1>
</body>
</html>
```

- Le code appelle un fichier séparé
- Le HTML est responsable du contenu
- Le CSS est responsable du design
- Le JS est responsable des interactions

Code exécuté à la fin du HTML

```
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <link rel="stylesheet" href="css/styles.css" type="text/css" />
  </head>
  <body>
    <h1>Hello World</h1>
  </body>
  <script type="text/javascript">
    console.log("Weather is not marvelous at all today");
    alert("Hey, what did you expect ?");
  </script>
</html>
```

- La balise `<script/>` se place n'importe où
- L'affichage de la page n'est pas caché
- Le JS ne devrait pas caché du texte après coup

Utilisation simple des fonctions

```
function doMath(a,b){  
    var x = 2 + a;  
    return x*b;  
}  
var result = doMath(2,3);  
console.log(result);
```

- Le type de retour n'est pas déclaré
- La fonction peut ne rien renvoyer
- Les types des paramètres ne sont pas déclarés

Paramètres facultatifs

```
function doMath(a,b){  
    var x = 2 + b;  
    return x*a;  
}  
var result = doMath(2,3,4); // 4 is ignored  
var result = doOtherMath(2); // Ouch !
```

- Ecrire une boucle calculant $5!$
- Ecrire une fonction calculant $x!$

Les types en Javascript

Les types

- Numériques
- String
- Booléen
- Tableau
- Objets
- Fonctions (voir plus loin)

Ainsi que l'éléments `undefined`. `null` est un des éléments particulier d'objet.

Les Strings

- Chaines de caractères
- Il n'existe pas de `char`
- Quelques fonctions associées

```
var str = "Hello World";
str.indexOf("llo");
str.search(/Wo/);
str.toLowerCase();
```

Exercice

- Ecrire une notice de ces 3 fonctions avec des exemples.
- Comment savoir si une string commence par une lettre ? Donner un exemple

Les numériques

- Les entiers se mélangent aux floats
- Théoriquement, il y a des problèmes de virgule

```
var x = 8;  
var y = 0.0000001;  
var z = y+x-y;  
// > z ?
```

Quelques fonctions

```
var str = "4"  
var x = parseInt(str);  
str = "4.23"  
var y = parseFloat(str);
```

Les booléens

Facile : `true` != `false`

Problème : les cas d'interprétations

Egalité simple

Trier entre true et false

```
true == "true"  
true == 0  
false == 0  
true == 1  
true == "hello"  
true == "1"  
"1" == 1
```

Conclusion ?

Egalité triple

```
true === "true"  
true === 0  
false === 0  
true === 1  
true === "hello"  
true === "1"  
"1" === 1
```

Conclusion ?

Les objets

```
var x = {}
var john = {
    name : "John Doe",
    id : 12,
    admin : true,
    friends : [{name:"Jamie", id:12, admin:false}, {...}, {...}]
}
console.log(john.name)
```

Objet composé

```
var x = {  
    user : {  
        name : "John Doe",  
        id : 12,  
        admin : true,  
        house:{  
            address : "12 Abbey Road, London",  
            id : 42  
        }  
    }  
}  
x.user.house.address == "12 Abbey Road, London";  
x.user.house.address === "12 Abbey Road, London"
```

Modification dynamique

```
var x = {text:"hello"}  
console.log(x.text);  
x.price = 2.24;  
console.log (x);  
x.editor = {name:"Gallimard", id:25};  
console.log (x);  
delete x.price  
console.log (x);
```

Langage dynamique

- Javascript est un langage dynamique
- Les attributs d'un objets sont non prévisibles
- Le compilateur détecte peu d'erreurs
- Les IDE aident peu
- Les développeurs peuvent être très créatifs

Egalité et pointeurs

- Les pointeurs posent rarement problèmes
- Mais les quelques problèmes sont vicieux !

```
var book = {text:"hello"}  
var y = {name:"Gallimard", id:25};  
book.editor = y;  
y.name = "Perlinpinpin";  
console.log(book.editor.name)
```

Clés

```
var strange = {  
    "id" : 3,  
    "---'!e'sik": "strange key !"  
}  
console.log(strange.---'!e'sik);  
console.log(strange[---'!e'sik]);
```

Les Tableaux

La déclaration est assez simple

```
var array = [];
var table = ["one", 2, "three", {x:4}, null, undefined];
typeof table; // --> object !!!
var x = table[3];
var y = table[12];
var z = what[2];
var u = table[-1];
//Assignment
table[2] = 35
table[-1]=12;
u = table[-1];
```

- En PHP, les objets sont des tableaux associatifs
- En Javascripts, les tableaux sont des objets ordonnés ou inversement

Quelques méthodes sur les tableaux

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.toString();
fruits.valueOf();
fruits.sort();
fruits.push("Coconut");
var coconut = fruits.pop();
var apple = fruits.shift();
fruits.unshift("Tangerine");
console.log(fruits.length); //property
```

Significations

- `toString()` / `valueOf`:
- `sort()`:
- `push(x)`:
- `pop()`:
- `shift()`:
- `unshift()`:
- `slice()`/`splice()`:

L'opérateur `typeof`

- `typeof` donne le type d'une variable
- ... avec quelques surprises

```
var x=null;
typeof x; //=> object
typeof ["hello", "world"];
typeof "Hello World"
typeof new String ("Hello World");
```

La programmation fonctionnelle

Définition

- fonction : entrées => sortie
 - entrées non modifiée
- S'oppose au modèle de programmation impérative
 - changement d'état des variables

Avantages

- Chaque fonction est facilement testable
- Chaque fonction est facilement réutilisable

Défaut

- Vite méli-mélo de fonction
- Un programme a toujours des effets de bord
 - sinon il ne fait rien

La programmation fonctionnelle en javascript

- Une fonction est une variable comme les autres
 - First citizen class
- Seule *feature* possible en quelques mois
- Mix Fonctionnel/Impératif
- Popularisé par les **callback** ajax

Exemples

```
function calcul(a,b, operation){  
    return operation(a,b);  
}  
var add = function(a,b){  
    return a + b;  
}  
var mult = function(a,b){  
    return a * b;  
}  
var result = calcul(2,5, add);  
console.log(result);
```

Ecriture des fonctions

```
function calcul(a,b, operation){  
    return operation(a,b);  
}
```

versus

```
var calcul = function (a,b, operation){  
    return operation(a,b);  
}
```

versus

```
var calcul = function whatever (a,b, operation){  
    return operation(a,b);  
}
```

Ecriture d'un forEach

```
function foreach(array, operation){  
    for (var i = 0 ; i < array.length ; i++){  
        var element = array[i];  
        operation( element, i );  
    }  
}  
var fruits = ["Tangerine", "Banana", "Orange", "Apple", "Mango", "Coconut"];  
foreach(fruits, function(element, index){  
    console.log("My fruit "+(index+1)+" is named : "+element);  
});
```

Alternative

Le deuxième argument de `foreach` est une fonction

```
foreach(fruits, function(element, index){  
    console.log("My fruit "+(index+1)+" is named : "+element);  
});
```

On aurait pu écrire aussi :

```
var closure = function(element, index){  
    console.log("My fruit "+(index+1)+" is named : "+element);  
}  
foreach(fruits, closure);
```

Exercice

- Afficher la liste des titles des topics
- Trier les fruits par nombre de lettres
- Trier les topics par nombre de commentaires

Les closures

- les closures sont des variables de type fonction
- Les closures en Javascript ont accès au contexte

```
var name = "Nicolas";
var closure = function(element, index){
    console.log(name+ " eats "+element);
}
foreach(fruits, closure);
```

- Cela peut avoir des conséquences de performances
- Il vaut mieux se limiter à l'accès de primitives

Retour d'une fonction

Une fonction peut renvoyer une fonction

```
function getSize(user){  
    var sizeUser = function(i, user){  
        return i + user.name.length  
    }  
    var sizeAdmin = function(i, admin){  
        return i + admin.statement.length  
    }  
    return user.admin ? sizeAdmin : sizeUser;  
}  
foreach(users, function(user, index){  
    console.log("size is "+getSize(user)(user, index));  
})
```

Quelques fonctions ES5

array.forEach()

- Permet de parcourir un tableau
- La fonction forEach ne renvoie pas de valeur
- On n'a pas accès à l'index
- Création d'effets de bords

```
users.forEach( function(user){  
    user.name = user.name + '-changed'  
    console.log(user.name);  
});
```

array.map()

- Renvoit un nouveaux tableau de même taille
- Chaque élément d'origine est *mappé* vers un autre

```
var even = [0,2,4,6,8];
function addOne(number){ return number +1 };
var odd = even.map(addOne); // [1,3,5,7,9]
```

Objectif possible map()

- Evite l'effet de bord des `forEach()`

```
var newUsers = users.map( function(user){  
  var newUser = clone(user);  
  newUser.name = newUser.name + '-changed'  
  return newUser;  
});
```

- Problème du clonage :
 - library
 - performance
- immutable.js est la solution adoptée

array.filter()

- Renvoit un nouveau tableau plus petit
- Récupère uniquement les éléments voulus

```
var numbers = [0,1,2,6,8,11];
function even(number){ return number%2 == 0};
var odd = numbers.filter(even); // [0,2,6,8]
```

Underscore, lodash

- Underscore est une bibliothèque fonctionnelle
- Plus téléchargée que jQuery sur npm !
- Ecrit par Jeremy Ashkenas
 - CoffeeScript
 - Backbone JS
- Copiée, augmentée par lodash

Exemple

Récupérer les contenus des Commentaires

```
var _ = require("underscore");
var topics = require("./topics").topics;
var users = require("./users").topics;
var result = _.map(topics, function(topic){
  var contents = [];
  _.each(topic.comments, function(comment){
    contents.push(comment.content);
  })
  return contents;
});
console.log(result)
```

Ou encore

```
var result = _.map(topics, function(topic){
  var contents = [];
  _.each(topic.comments, function(comment){
    contents.push(comment.content);
  })
  return contents;
});
```

se remplace par :

```
var result = _.map(topics, function(topic){
  return _.pluck(topic.comments, "content")
});
```

Exercice

Avec le minimum de code :

- Récupérer les commentaires écrits par Penny et les trier par taille
 - map, sort
- Faire un objet triant les contenus commentaires par user
 - groupBy

```
{  
    nicolas : ["Hello", "Blabla"],  
    sheldon : [..., ...]  
}
```

Le Prototype

La programmation prototypée en JavaScript

Objectifs

- JavaScript n'a pas de notion de class comme on a sur Java ou C#
- Pas de *class* -> on crée un objet à partir d'un autre
 - Plus libre et rapide avec JavaScript
- La notion d'objet et l'héritage existe ("*Prototypal inheritance*")
- Une fonction est un objet, un tableau est un objet
- Le **Prototype** est un design pattern

Exemple

```
// Comment commence par une majuscule par convention
// pour dire que c'est un constructeur
function Comment(subjectId, content){
    this.subjectId = subjectId;
    this.content = content;
}
var comment = new Comment(1,'First comment!');
```

Prototype d'une fonction

- Toutes les fonctions ont une propriété *prototype*
- Toutes les propriétés prototypes sont ajouté à la fonction
- Chercher les propriétés/méthodes dans l'objet puis dans les prototypes après
- Performance : prototype > propriété directe

```
function Comment(){}
var c = new Comment();
Comment.prototype.setContent = function(content){
    this.comment = comment;
};
c.setContent('First comment');
alert(c.comment);
```

Augmentation d'objet

```
var anonymousComment = new Comment(0,'First comment');
//ajout d'anonymous
anonymousComment.isAnonymous = true;
alert(anonymousComment.subjectId);
var authenticatedComment = new Comment(1,'Second comment');
authenticatedComment.isAnonymous = false;
authenticatedComment.userName = 'Robusta';
alert(authenticatedComment.SubjectId);
```

Héritage prototypé

```
function Comment(){}
Comment.prototype.subjectId = 0;
Comment.prototype.content = "";
function AuthenticatedComment(userName){
    this.userName = userName;
};
AuthenticatedComment.prototype = new Comment();
AuthenticatedComment.prototype.isAnonymous = false;
var comment = new AuthenticatedComment('jean');
console.log(comment.subjectId);
console.log(comment.content);
console.log(comment.userName);
console.log(comment.isAnonymous);
```

Le rôle de l'opérateur new

```
Forum = function() {this.admin = 'nicolas';};  
// Just a function  
Forum.prototype.copyright = 'Robusta Code';  
// like all functions, Forum has an accessible prototype property  
club = new Forum();  
// 3 things just happened  
// A new, empty object {} was created called club.  
// The [[prototype]] property of club was set to a copy of the prototype property  
// of Forum. The Forum function was executed, with club in place of "this"  
// so club.admin was set to 'nicolas'  
club.admin; // returns 'nicolas'  
club.copyright;  
// club doesn't have a property called 'copyright',  
// Its [[prototype]] is the same as Forum.prototype. NOT A COPY !  
// Forum.prototype has a property called 'copyright' with value 'Robusta Code'  
// returns 'Robusta Code'
```

Parenthèses Optionnelles

```
//identiques
var club = new Forum();
var club = new Forum;
var comment = new Backbone.Model
//Attention aux mauvaises habitudes. Non identiques !
var club = Forum();
var club = Forum;
```

InstanceOf

```
comment instanceof AuthenticatedComment; //true  
comment instanceof Comment; //true  
comment instanceof Object; //true
```

Créer des *objets* à partir des données brut

```
function Forum(name, admin){  
    this.name = name,  
    this.admin = admin  
}  
Forum.prototype = {  
    addTopic : function(title, user){...},  
    removeTopic : function(id){...},  
    toString : function(){...}  
}  
function Topic(title){  
    this.title = title  
}  
Topic.prototype = {  
    addComment : function(content, user){...},  
    removeComment : function(id){...},  
    toString : function(){...}  
}
```

Autres objets : User, Admin, Comment, Like...

Interpretation en Javascript

Undefined, null,

- `undefined` : variable non définie
- `null`: variable volontairement mise à `null`
- `0` : valeur numérique volontairement mise à `0`
- `false` : valeur booléene mise volontairement mise à `false`

Exemple de valeur undefined

```
function doMath(a,b){  
    console.log(b);  
    var x = 2 + b;  
    return x*a;  
}  
doMath(1); //b a donc la valeur undefined  
var user={  
    name:"John",  
    id:1,  
    value:null  
}  
//user.name == "John"  
//user.age == undefined  
//user.value == null
```

Exemple compliqué

- Ouvrir la console JS
- taper `myVar` + Entrée
- taper `x={}` + Entrée
- taper `x.myVar` + Entrée
- taper `var myOtherVar` + Entrée

Dans certains cas, l'interpréteur peut donc s'interompre plutôt que de comprendre *undefined*

Tester si une variable existe

- `myNewVar + Entrée` => Error
- `typeof myNewVar` => `undefined`

```
if(typeof(myNewVar !== undefined){  
    //do stuff here  
}else{  
    alert('something is wrong');  
}
```

Jquery

Utilité du Framework

- La **manipulation du DOM** est fastidieuse.

Pour faciliter la tache, on utilise un Framework :

- jQuery
- Zepto
- Dojo
- ...

Autre utilisation : **animation, Ajax**

Principe de base

- Filtrer : *selectors*
- Agir sur le(s) élément(s)
 - changer le contenu
 - gerer les événements des éléments
 - changer la structure
- jQuery est-il une monade ?

Syntaxe de base

```
$('div').addClass('container');  
$('div.content1').slideUp();  
$('div.content2').slideDown();  
jQuery('div').addClass('container');  
$('#comments').plugins1(); // importance du nombre de plugins disponible
```

Remarque : Object jQuery => `$` ou `jQuery`

Sélecteurs jQuery

- `$('h1')` : tous les `<H1>`
- `$('#container')` : l'élément avec `id = "container"`
- `$('div#container')` : l'élément avec `id = "container"` si `<div>`
- `$('div.container')` : tous les divs avec la classe *container*
- `$('div#container p')` : tous les `<p>` dans l'élément *container*
- `$('div p')` : tous les `<p>` inclus dans un `<div>`

Google : CSS Diner

Performances

Plutôt que

```
var x = $("#container p");
var y = $("#container div");
```

Mieux vaux :

```
var container = $("#container");
var x = container.find("p");
var y = container.find("div");
```

L'état de l'art suggère de sélectionner uniquement à partir des classes.

Manipulation du DOM

```
$('input#age').val()  
$('input#age').val("18")  
$('div#content').append('Hello')  
$('div#content').prepend('Hello')  
$('div.desktop').remove()  
var desktop = $('div#desktop').detach()  
$('div.desktop').hide()
```

Gestion des Evénements

```
$(document).ready(function(){
    //Excuter le scrit après le chargement de la page
});
//old school
$('div#container').click(function(e){
    //TODO
});
//Much Backbone style
$('div#container').on("click", function(e){
    //TODO
});
// delegate : une fonction pour de multiples éléments
$("div#container").delegate("click", "li", function() {
    $( this ).toggleClass("selected");
});
```

StopPropagation et preventDefault

```
$( ".send" ).on( "mouseover", function( event ) {  
    event.stopPropagation();  
    //...  
});  
$( "a.action" ).click(function( event ) {  
    event.preventDefault();  
    //...  
});
```

Ajax

- `$.ajax()`
- `$.get()`
- `$.post()`
- `$.getJSON()`

\$.ajax()

```
$.ajax({  
    type:"GET"  
    url: "users/all",  
    data: "{user:2, name:'john'}", //?user=2&name=john for GET  
    success: function(data){....},  
    error: function(xhr, status, error){....},  
    dataType: "json"  
});
```

Ajax Promise Style

```
//promise style
$.get("/forum/1/users")
  .then(function(users) { return data.pluck('name') })
  .then(function(userNames) { displayOnScreen(userNames) })
  .catch(function() { alert("error"); })
  .finally(function() { alert("job complete"); });
```

Chaining

```
$('div#container').css("color","blue").slideDown();
```

Problème majeur de jQuery

- query le DOM... Pas performant
- Insertion de donnée dans le DOM : data-id, etc...
- Création du DOM au fil de l'eau
 - Utilisation de Mustache/Handlebar
- Code peu structuré, pas objet

Ceci dit, jQuery, c'est génial !

Javascript Build

Objectifs

- Gestion des dépendances : jQuery, angular...
- Eviter l'import de 50 `<script>` dans une page
 - performance
 - ordre des scripts
- Faire des opérations courantes
 - déplacer des fichiers
 - concaténer, minifier
- Résoudre plusieurs problèmes courants
 - Transpiler sass, coffeescript, typescript, haml
 - chargement de template
 - Linters, lancement de tests

Node JS

- Execute du code Javascript
 - node myFile.js
 - utilise V8
- Créer un serveur d'application
 - Node
 - Connect : serveur
 - Express :framework
 - Mongoose : ORM
 - MongoDB

Gestion des dépendances

NPM

- NodeJS est packagé avec NPM
- Nécessite un fichier `package.json`
- NPM gère des modules de type CommonJS
 - `npm install myModule`
 - `require("myModule")`
- NPM regroupe les dépendances d'un projet dans `package.json`
 - `npm install --save myModule`

Bower

- NPM gère les dépendances du serveur
- Creation d'un fichier `bower.json` équivalent
- Bower gère les dépendances du navigateur
 - jamais du server node
- Obsolète aujourd'hui

Graphe de dépendance

Les outils modernes :

- Construisent un graphe des librairies utilisées
- Piochent dans `package.json`

```
// Building a graph with JS or HTML
var userModule = require('./user-module');
var template = require('./customer-optional-template.html');
function DisplayUserModule(){
    // Module code
}
//Creating a module for current file
module.exports = DisplayUserModule;
```

L'ecosystème

Modules

- AMD : RequireJS ; obsolète
- CommonJS : NodeJS, Browserify sur browser
- ES 2015
- SystemJS : AMD + CommonJS + ES 2015
- ES2015 est disponible sur Node JS 4
 - Common JS tend aussi à être obsolète

Exemple de modules : Common JS

Création du premier module :

```
// salute.js
var MySalute = "Hello";
module.exports = MySalute;
```

Création d'un deuxième module appelant le premier :

```
// world.js
var MySalute = require("./salute");
var Result = MySalute + " world!";
module.exports = Result;
```

!info : Utilisation de mots *non-clés* `module` et `exports`

Exemple de modules : ES 2015

Création du premier module :

```
// salute.js
var MySalute = "Hello";
// 'default' means we export only one object
export default MySalute;
```

Création d'un deuxième module appelant le premier :

```
// world.js
import MySalute from './salute';
var Result = MySalute + " world!";
var OtherResult = MySalute + " guy!";
export Result;
export OtherResult;
```

Transpilers

- CoffeeScript : mode Ruby ; précurseur
- TypeScript :
 - typé, proche de Java
 - Surcouche de ES2015 (dont les modules)
- Sass, Less :
 - Simplifie les CSS : variables, nesting
 - Sass est top pour le responsive

Build

- Grunt / Gulp : automatise les tâches (trancile, minification, tests)
 - grunt : declaratif (json)
 - gulp : pure code javascript
- Webpack : Alternative à gulp + browserify
- JSPM + SystemJS : Alternative à npm + gulp + browserify

exemple Gulp

```
// gulpfile.js
var gulp = require('gulp');
var tsify = require('tsify'); //... need all npm dev-dependencies
function bundle() {
  browserify({debug:true})
    .transform(stringify, {
      appliesTo: { includeExtensions: ['.hjs', '.html', '.txt'] }
    })
    .add('index.ts')
    .plugin(tsify)
    .bundle()
    .on('error', function (error) { console.error(error.toString()); })
    .pipe(exorcist('./dist/app-build.js.map'))
    .pipe(vinylSourceStream('./dist/app-build.js'))
    .pipe(gulp.dest('./'));
}
gulp.task('by', bundle);
```

Execute : '\$> gulp by

Autres outils

- Jasmine, Karma : outils de tests unitaires automatique
- Selenium, Protractor : tests d'intégration
- PhoneGap : HTML5 sur mobile
- JS Lint & JS Hint : Equivalent de checkstyle

Les frameworks de haut-niveau

Backbone



BACKBONE.JS

- Crée par Jeremy Ashkenas (CoffeeScript, Underscore)
- Très petit (6KB) et compréhensible
- Necessite beaucoup de code, moins productif
- Mais beaucoup plus souple
- Fonctionne avec jQuery

Angular



- Crée par Miško Hevery et Vojta Jina
- Plus gros
- Plusieurs pallier d'apprentissage
- Très productif ... si on n'est pas bloqué
- Peut fonctionner avec jQuery
- Angular 1 ou Angular 2 ?

React

- Très performant
- Crée et utilisé en production par Facebook
- Uniquement partie View
- Se simplifie/complexifie avec Flux/Redux



ember basics

PROJECT STRUCTURE

- Gros projet
- Semble compliqué
- Très *framework*

ExtJS / Sencha Touch



- Existe depuis 2006
- Beaucoup de widgets performants
- Maintenant avec une orientation Mobile

Google : *ext js kitchen* Google : *sencha touch kitchen*

Les nouvelles fonctionnalités Javascript HTML5

Canvas

- Possibilité de dessiner
- Utilisation du GPU
- Canvas est une API de bas niveau **non normée**
 - existe en C, Java, Python...
- Image matricielle, évite les blocks HTML
- Activation avec :

```
var canvas = document.getElementById("canvas");
var context = canvas.getContext("2d");
```

Canvas versus SVG

- SVG est également un standard (pré-HTML5)
- SVG est vectoriel
- Non lié au Javascript
 - Mais utilisable avec Raphaël

Google : Chrome Angry Birds

La 3D avec WebGL et Three.js

- WebGL, c'est **compliqué**
- ... Mais sans doute pas plus compliqué qu'autre chose
- Similaire à Canvas pour la 3D
- Nécessite le langage `x-shader`
- Utilise le GPU

Exemple

```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
    }
</script>
```

cf demos : svg ; rectangle ; simple-rectangle

Three.JS

- Three.js est une API de haut niveau
- Basée sur WebGL
- Menée par mrdoob
- Code plus compréhensible

Three.js : code

```
function init() {
    scene = new THREE.Scene();
    camera = new THREE.PerspectiveCamera( 75, window.innerWidth / window.innerHeight, 1, 10000 );
    camera.position.z = 1000;
    geometry = new THREE.BoxGeometry( 200, 200, 200 );
    material = new THREE.MeshBasicMaterial( { color: 0xff0000, wireframe: true } );
    mesh = new THREE.Mesh( geometry, material );
    scene.add( mesh );
    renderer = new THREE.WebGLRenderer();
    renderer.setSize( window.innerWidth, window.innerHeight );
    document.body.appendChild( renderer.domElement );
}
```

Three.js : Features

- Lumières
- Animation
- Textures

Avec algorithme :

- Ne pas calculer les faces cachées
- Import Autocad/Catia

Demo : <http://helloracer.com/>

Le stockage sur le navigateur

- Cookie
- Web Storage : Local Storage & Session Storage
- WebSQL : deprecated
- IndexedDB : l'avenir
- Application Cache : Applications HTML5 offline

Cookie

- Limité à 4ko
- Mal vu
- Mentions légales obligatoires
- Attention, les cookies ne fonctionnent pas sur un fichier sans serveur
- Utiliser le plugin cookie de jQuery

The EU Cookie Law

- Les sites web retenant des informations locales doivent avoir l'assentiment de l'utilisateur
- On ne peut pas se contenter d'une information
- Le LocalStorage ou l'utilisation de Flash est également concerné
- La loi est valable pour les ordinateurs, smartphones, tablettes, etc...
 - mais pas pour les application mobiles téléchargées...
- Tout mais pas l'indifférence...

Web Storage

- Très simple à utiliser
- Ne stocke pas les objets
- Local Storage : permanent
- Session Storage : s'en va avec la session
 - toutes fenêtres fermées
- 5 à 10 Mo avant une demande
- Clairement mieux que

IndexedDB

- Utilisation de clé primaire
- Utilisation d'index
- Storage de vrais objets JSON
- Pas de jointure entre collections
 - principe NoSQL
- 50 Mo de données

WebSQL

- Deprecated
- Basé sur SQLite
- Pas de gestion des transitions
 - Script SQL
- Jointures

Application Cache

- Système de fichier Manfiest
- Permet l'utilisation offline d'une application
 - pas franchement un cache applicatif !
- Syntaxe très descriptive
- Comme pour le mobile : Complexe !!!

Application Cache : Exemple simple

index.html :

```
<html manifest="http://www.example.com/example.mf">  
</html>
```

/example.mf :

```
CACHE MANIFEST  
index.html  
stylesheet.css  
images/logo.png  
scripts/main.js  
http://cdn.example.com/scripts/main.js
```

Application Cache : Exemple détaillé

```
CACHE MANIFEST
# Explicitly cached 'master entries'.
CACHE:
/favicon.ico
index.html
stylesheet.css
images/logo.png
scripts/main.js
# By default we always go online
NETWORK:
*
# static.html will be served if main.php is inaccessible
# offline.jpg will be served in place of all images in images/large/
FALLBACK:
/main.php /static.html
images/large/ images/offline.jpg
```

Résumé

- WebSQL : ne pas utiliser !
- Cookie : limité
- LocalStorage : simple
- IndexedDB : complexe, performant, gros volume
- Application cache : si le besoin est réel

Communication avec WebRTC

- Standard proposé par Google
- Validé par le W3C
- S'oppose à Skype, WebEx.. et Hangouts
- Necessite un serveur, pas du P2P
- Pas disponible ni prévu sur IE ni Safari
- Demo : opentokrte

Les WebSockets

- Communication *Real time*
- Le navigateur fait office de serveur
- Notification *push*
- Support par IE 10+, fallback sur *Long polling*
- Exemple : Trello, Slack

Les Web Workers

- Equivalent du Multithreading
- *Browser feature* accessible via Javascript
- Utile pour les travaux lourds
 - Arranger une liste d'objets au demmarage de l'appli
- Supporté par IE 10+
- Permet de dialoguer avec un contexte indépendant
- Les objets transitent sans copie
- Difficile à debugguer
- Plutôt simple et efficace

Appel

index.html :

```
var myWorker = new Worker("webworker.js");
myWorker.addEventListener("message", function (event) {
    console.log("worker says : ",event);
}, false);
myWorker.postMessage(100000); // start the worker.
```

webworker.js :

```
self.addEventListener('message', function(e) {
    var max = e.data;
    for (var i = 0 ;i < max ; i++) { users...}
    self.postMessage(users);
}, false);
```