



formation.tech

# Formation JavaScript

Romain Bohdanowicz

Twitter : @bioub - <https://github.com/bioub>

<http://formation.tech/>



**formation.tech**

# Introduction

# Présentations



- Romain Bohdanowicz  
Ingénieur EFREI 2008, spécialité en Ingénierie Logicielle
- Expérience  
Formateur/Développeur Freelance depuis 2006  
Plus de 10 000 heures de formation animées
- Langages  
Expert : HTML / CSS / JavaScript / PHP / Java  
Notions : C / C++ / Objective-C / C# / Python / Bash / Batch
- Certifications  
PHP 5 / PHP 5.3 / PHP 5.5 / Zend Framework 1 / Node.js Application Developper
- Particularités  
Premier site web à 12 ans (HTML/JS/PHP), Triathlète à mes heures perdues
- Et vous ?  
Langages ? Expérience ? Utilité de cette formation ?



**formation.tech**

# JavaScript IDEs



# JavaScript IDEs - Webstorm

- Version orientée Web de IntelliJ IDEA de l'éditeur JetBrains  
<https://www.jetbrains.com/webstorm/>
- Licence : Commercial  
Licence entre 35 à 129 euros par an selon le profil et l'ancienneté.  
Version d'essai 30 jours.
- Plugins :  
Annuaire (642 en novembre 2016) : <https://plugins.jetbrains.com/webStorm>  
Langage de création : Java





# JavaScript IDEs - Webstorm

functionnal.js - Language - [~/www/Learning/JavaScript/Language]

Language > Array > functionnal.js

Project Structure

Language ~ /www/Learning/JavaScript/Language

- Array
  - functionnal.js
- ES5.1
- EventLoop
- Function
- Number
- Objet
- Promesse
- addressbook.json
- arrays.js
- closure.js
- conversions.js
- eval.js
- exceptions.js
- existing\_var.js
- functions.js
- json.js
- loops.js
- newObject.js
- object\_advanced.js
- reference.html
- reference.js
- regexp.js
- strict.js

Run functionnal.js

/usr/local/bin/node /Users/romain/www/Learning/JavaScript/Language/Array/functionnal.js  
ERIC  
JEAN

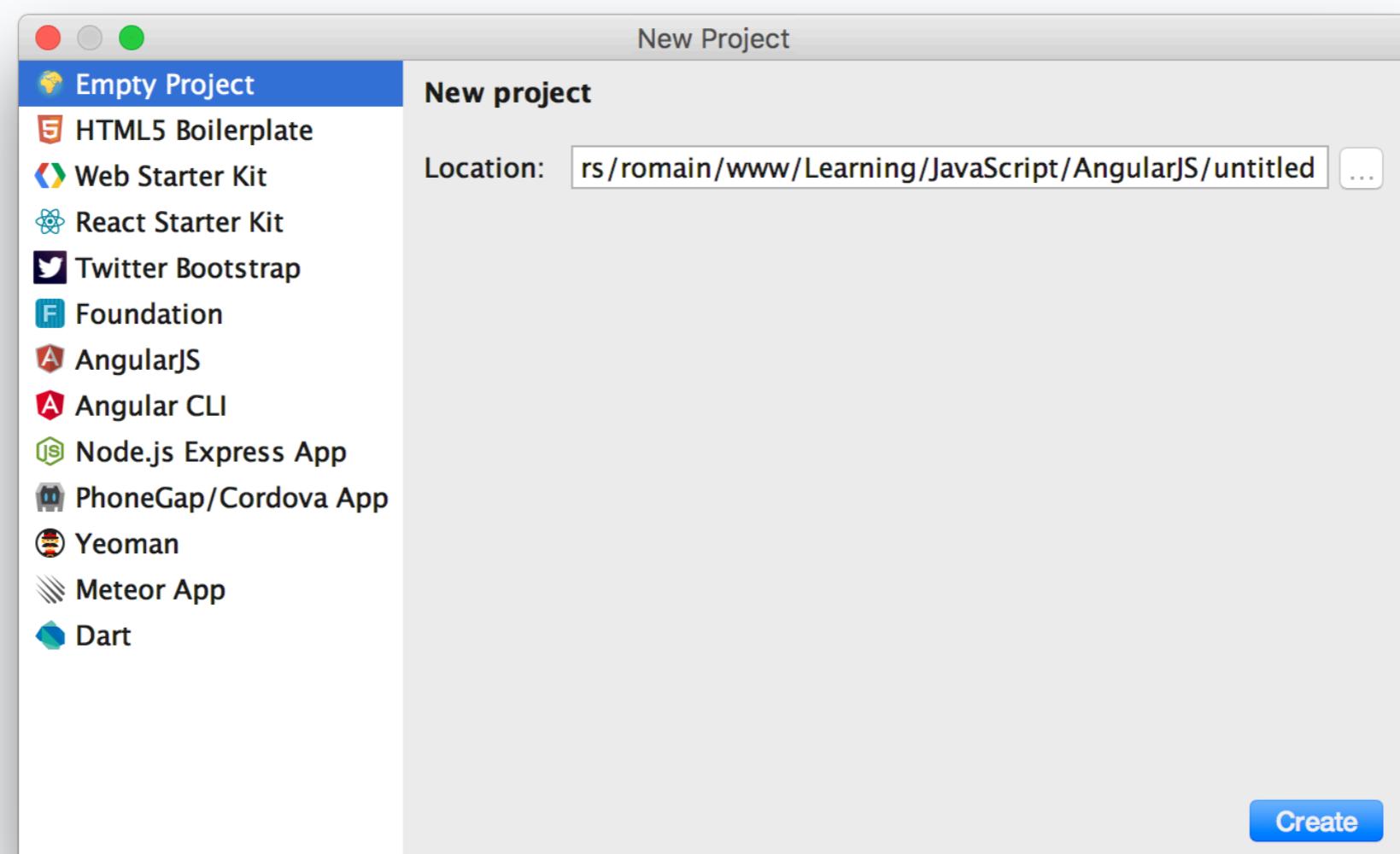
Process finished with exit code 0

4: Run 6: TODO Terminal Event Log

11:1 LF UTF-8

```
1 var firstNames = ['Romain', 'Jean', 'Eric'];
2
3 firstNames.filter((firstName) => firstName.length === 4)
4   .map((firstName) => firstName.toUpperCase())
5   .sort()
6   .forEach((firstName) => console.log(firstName));
7
8 // Outputs :
9 // ERIC
10 // JEAN
```

# JavaScript IDEs - Webstorm



# JavaScript IDEs - Webstorm



Run/Debug Configurations

Configuration Browser / Live Edit V8 Profiling

Node interpreter: /usr/local/bin/node (Project) 7.0.0

Node parameters:

Working directory:

JavaScript file:

Application parameters:

Environment variables:

Before launch: Activate tool window

There are no tasks to run before launch

+ - ⚪

Show this page  Activate tool window

Cancel Apply OK

The screenshot shows the 'Run/Debug Configurations' dialog in Webstorm. The left sidebar lists various configuration types under 'Node.js' and 'Defaults'. The 'Node.js' section is currently selected. The main panel displays fields for setting up a Node.js run configuration, including the node interpreter path, parameters, working directory, JavaScript file, application parameters, and environment variables. A 'Before launch' section indicates there are no tasks to run before launch. At the bottom, there are checkboxes for 'Show this page' and 'Activate tool window', along with standard 'Cancel', 'Apply', and 'OK' buttons.

# JavaScript IDEs - Webstorm



Preferences

Languages & Frameworks > JavaScript > Libraries For current project

Libraries

Enabled	Name	Type
<input type="checkbox"/>	angular-cookie-DefinitelyTyped	Global
<input type="checkbox"/>	express-DefinitelyTyped	Global
<input checked="" type="checkbox"/>	ECMAScript 6	Predefined
<input checked="" type="checkbox"/>	HTML	Predefined
<input checked="" type="checkbox"/>	HTML5 / ECMAScript 5	Predefined
<input type="checkbox"/>	WebGL	Predefined

Add... Edit... Remove Download... Manage Scopes...

Cancel Apply OK

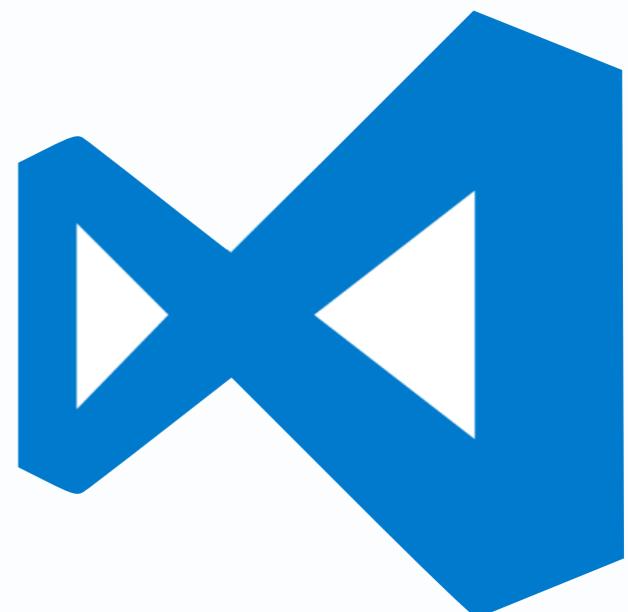
Search

- > Editor
- Plugins
- > Version Control
- Directories
- > Build, Execution, Deployment
- > Languages & Frameworks
  - > JavaScript
    - Libraries
    - > Code Quality Tools
      - JSLint
      - JSHint
      - Closure Linter
      - JSCS
      - ESLint
    - Templates
    - Bower
    - Yeoman
    - PhoneGap/Cordova
    - Meteor
  - > Schemas and DTDs
    - Compass
    - Dart
  - > Markdown
  - Node.js and NPM

# JavaScript IDEs - Visual Studio Code



- IDE créé par Microsoft, tourne sous Electron (Chromium + Node.js)  
<http://code.visualstudio.com/>
- Licence : MIT  
La licence open-source la plus permissive
- Plugins :  
Annuaire (1867 en novembre 2016) : <https://marketplace.visualstudio.com/VSCodium>  
Langage de création : JavaScript sous Node.js
- Documentation  
<https://code.visualstudio.com/docs>





# JavaScript IDEs - Visual Studio Code

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure with files like `about.module.ts`, `parse5-adapter.js`, `api.ts`, `app.node.module.ts`, `app.browser.module.ts`, `home.module.ts`, `index.js`, `client.ts`, and `index.html`.
- Code Editor (Center):** The active file is `about.module.ts`. The code is as follows:

```
1 import { Title } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3
4 import { AboutComponent } from './about.component';
5 import { AboutRoutingModule } from './about-routing.module';
6
7 @NgModule({
8   imports: [
9     AboutRoutingModule
10 ],
11 declarations: [
12   AboutComponent
13 ],
14 providers: [
15   Title
16 ],
17 })
18 export class AboutModule { }
```

- Terminal (Bottom):** Shows the command line interface with the following output:

```
master* 29↓ 0↑  x 2 ▲ 0  {..}:14
```
- Status Bar (Bottom):** Displays information about the current file: Li 19, Col 1 Espaces : 2 UTF-8 LF TypeScript 😊



# JavaScript IDEs - Visual Studio Code

The screenshot shows a Visual Studio Code window with the following details:

- Title Bar:** hello.js - VisualStudioCode
- Code Editor:** JS hello.js (line 3 is highlighted in yellow)

```
1 const hello = function () {  
2     let message = 'Hello';  
3     console.log(message);  
4 };  
5  
6 setTimeout(hello);
```
- Left Sidebar:**
  - DÉBOUGER (Debug) panel:
    - VARIABLES section: Local (message: "Hello")
    - PILE DES APPELS (Call Stack) section: EN PAUSE SUR POINT D'ARRÊT (Paused at breakpoint). Shows stack frames for hello, ontimeout, tryOnTimeout, and listOnTimeout.
    - POINTS D'ARRÊT (Breakpoints) section: Shows checkboxes for Toutes les exceptions, Exceptions interceptées (checked), and hello.js (checked).
- Output Panel:** SORTIE (Output) tab, ESLint section:

```
/usr/local/lib/node_modules/eslint/lib/api.js  
[Warn - 5:51:55 PM]  
No ESLint configuration (e.g. .eslintrc) found for  
file: hello.js  
File will not be validated. Consider running the  
'Create .eslintrc.json file' command.  
Alternatively you can disable ESLint for this  
workspace by executing the 'Disable ESLint for this  
workspace' command.
```
- Bottom Status Bar:** Li 3, Col 1 Espaces : 4 UTF-8 LF JavaScript ESLint ☺

# JavaScript IDEs - Atom



- IDE créé par Github, tourne sous Electron (Chromium + Node.js)  
<https://atom.io>
- Licence : MIT  
La licence open-source la plus permissive
- Plugins :  
Annuaire (5232 en novembre 2016) : <https://atom.io/packages>  
Langage de création : JavaScript sous Node.js  
Exemples : atom-ternjs, linter, JavaScript Snippets, autocomplete+, autoprefixer...)





# JavaScript IDEs - Atom

The screenshot shows the Atom code editor interface. The left sidebar displays the project structure:

- TestFlow (selected)
- .idea
- AmdLoader
- c
- Flow
  - UIKIT
    - Input
      - Button.js
      - ButtonModule.js
  - TestFlowWidget
    - Button.js
    - Button.ts
    - index.html
    - test.js
    - tsconfig.json
  - TodoDS
    - UI
      - .TodoDS.index.mk
      - index.html
      - initial\_ui.html
      - TodoDS.js
  - UIKIT
  - .flowconfig

The main editor area shows the content of the `index.html` file:

```
13 <!-- Application Metas End -->
14 <!-- Application Standalone emulation files -->
15     <link rel="stylesheet" href="../../UWA/assets/css/standalone.css" />
16     <script src="../../AmdLoader/AmdLoader.js"></script>
17     <script src="../../UWA/js/UWA_Standalone_Alone.js"></script>
18
19         <!-- UIKIT files -->
20     <link rel="stylesheet" href="../../UIKIT/UIKIT.css">
21     <script src="../../UIKIT/UIKIT.js"></script>
22
23         <!-- Application JS Start -->
24         <script>
25             /* global widget, require */
26             require(['DS/TodoDS/TodoDS'], function(main) {
27                 'use strict';
28
29                 var myWidget = {
30
31                     //The onLoad() function is the first one,
32                     //it will be triggered by widget "onLoad" event.
33                     onLoad: function() {
34
35                         // Replaces body contents
36                         //
37                         //widget.body.innerHTML= "Hello World";
38                         main(widget.body);
39
40                     };
41
42                     //The "onLoad" event is the very first event triggered when
43                     // the widget is fully loaded.
44                     widget.addEvent('onLoad', myWidget.onLoad);
45
46                 );
47                 </script>
48             </head>
49             <body>
```

At the bottom, the status bar shows: File 0 Project 0 ✓ No Issues TodoDS/index.html 1:1 ▲ 1 deprecation UTF-8 HTML 📁 1 update

# EditorConfig



- Permet de standardiser les configs des IDEs sur l'indentation et les retours à la ligne  
<http://editorconfig.org>
- Supporté par la plupart des IDE
- Il suffit de créer un fichier .editorconfig à la racine d'un projet

```
# EditorConfig is awesome: http://EditorConfig.org

# top-most EditorConfig file
root = true

# Unix-style newlines with a newline ending every file
[*]
end_of_line = lf
insert_final_newline = true
charset = utf-8
indent_style = space
indent_size = 4

# HTML + JS files
[*.{html,js}]
indent_size = 2
```



**formation.tech**

# JavaScript (ECMAScript 3)

# JavaScript - Introduction



- Langage créé en 1995 par Netscape
- Objectif : permettre le développement de scripts légers qui s'exécutent une fois le chargement de la page terminé
- Exemples de l'époque :
  - Valider un formulaire
  - Permettre du rollover
- Netscape ayant un partenariat avec Sun, nomma le langage JavaScript pour qu'il soit vu comme le petit frère de Java (dont il est inspiré syntaxiquement)
- Fin 1995 Microsoft introduit JScript dans Internet Explorer
- Une norme se créa en 1997 : ECMAScript

# JavaScript - ECMAScript



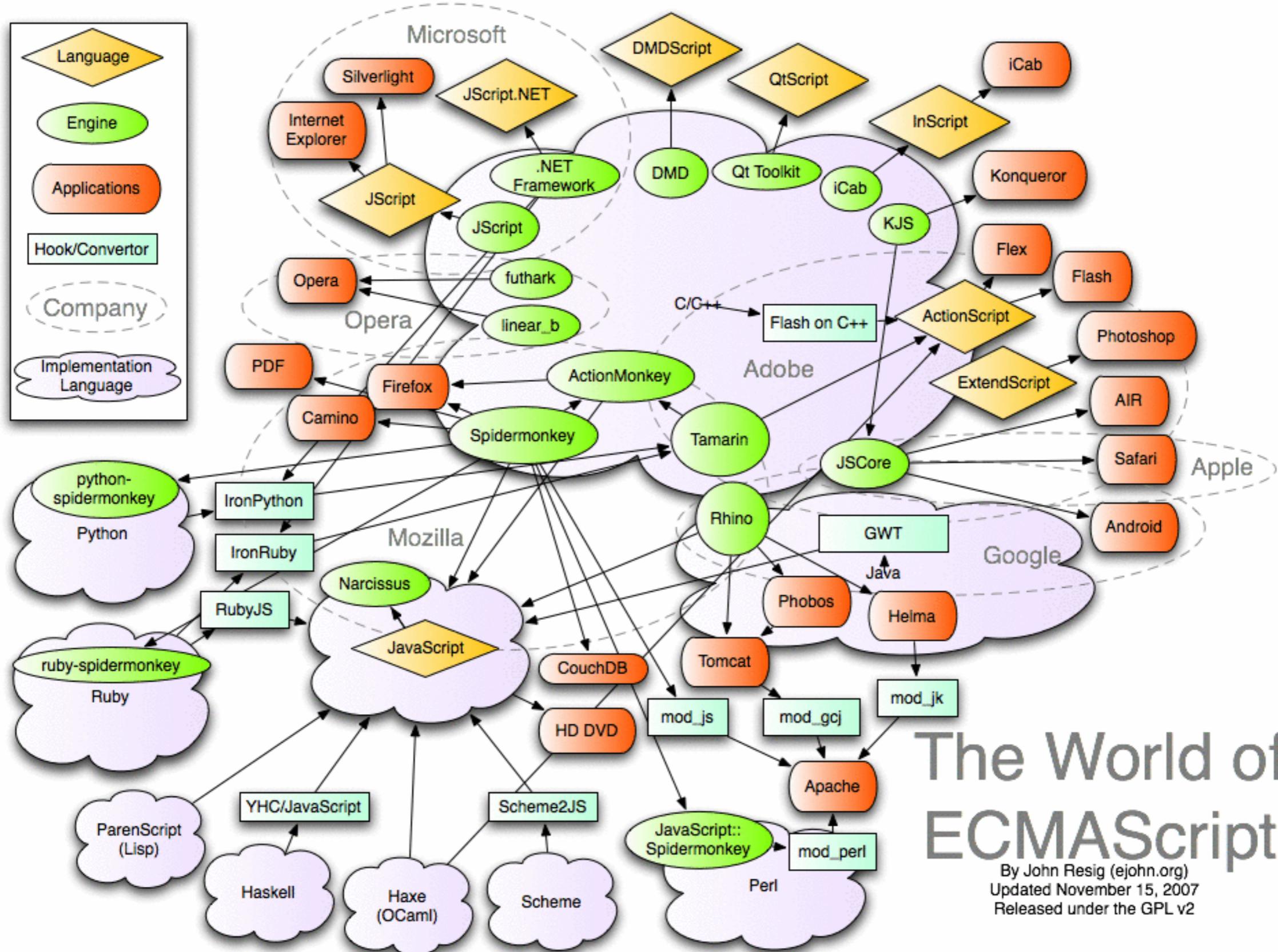
- JavaScript est une implémentation de la norme ECMAScript 262
- La norme la plus récente est ECMAScript 2019  
Aussi appelée ECMAScript 10 ou ES10 (juin 2019)  
<https://www.ecma-international.org/ecma-262/10.0/>
- Le langage a très fortement évolué avec ECMAScript 2015  
ECMAScript 6 / ES6 (juin 2015)  
<https://www.ecma-international.org/ecma-262/6.0/>
- Pour connaître la compatibilité des moteurs JS :  
<http://kangax.github.io/compat-table/>
- Compatibilité ES6  
Navigateur actuels (octobre 2016) ~ 90% d'ES6  
Node.js 6 ~ 90% d'ES6  
Internet Explorer 11 ~ 10% d'ES6

# JavaScript - Documentation



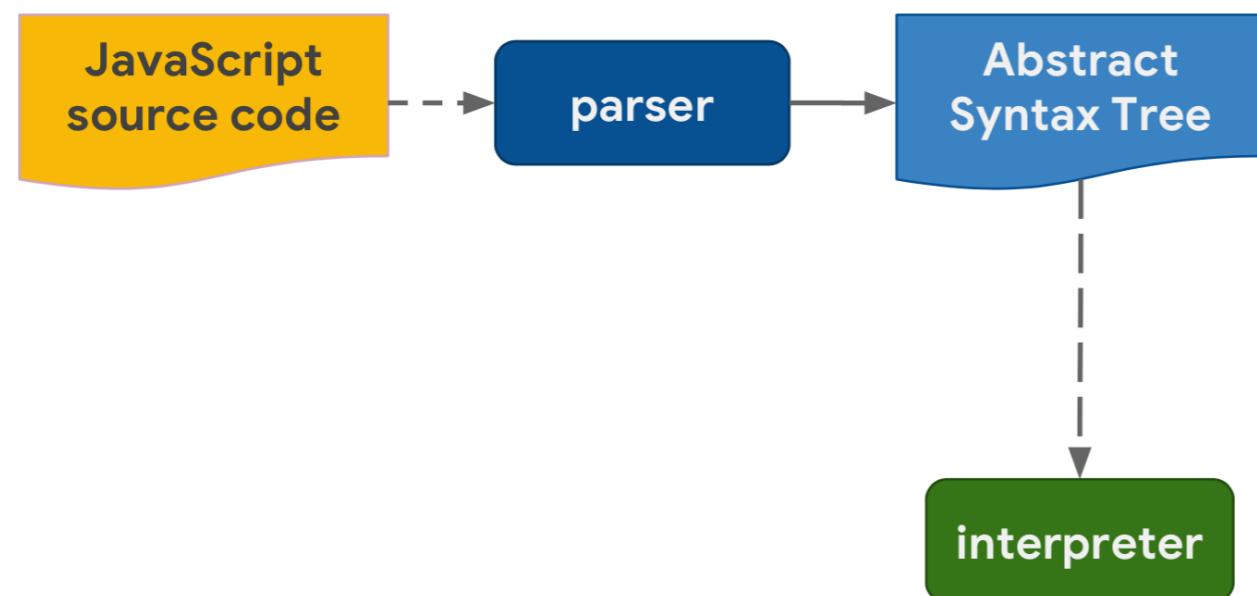
- La norme manque d'exemples et d'information sur les implémentations :  
<https://www.ecma-international.org/ecma-262/10.0/>
- Mozilla fournit une documentation open-source sur le langage JavaScript et sur les APIs Web (utiliser la version anglaise qui est plus à jour) :  
<https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- DevDocs permet de retrouver la documentation de Mozilla en mode hors-ligne  
<http://devdocs.io/javascript/>

# JavaScript - ECMAScript

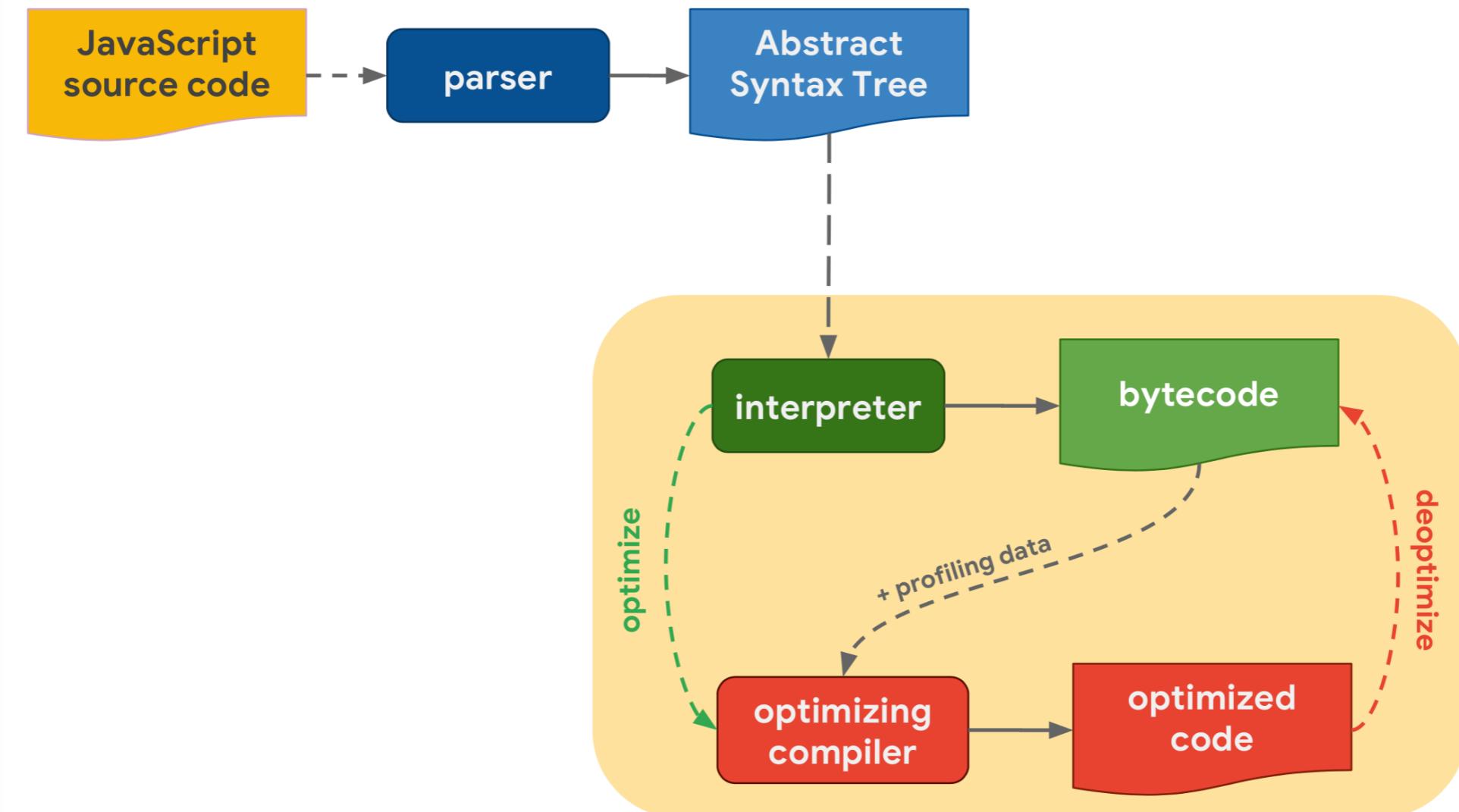




# JavaScript - Interprétation



# JavaScript - Compilation JIT



- <https://slidr.io/mathiasbynens/javascript-engines-the-good-parts>

# JavaScript - Syntaxe



- La syntaxe s'inspire de Java (lui même inspiré de C)
- JavaScript est sensible à la casse, attention aux majuscules/minuscules !
- Les instructions se terminent au choix par un point-virgule ou un retour à la ligne (même si les conventions incitent à l'utilisation du point-virgule)
- 3 types de commentaires
  - // le commentaire s'arrête à la fin de la ligne
  - /\* commentaire ouvrant/fermant \*/
  - /\*\* Documentation JSDoc -> <http://usejsdoc.org/> \*/



# JavaScript - Identifiants

- ▶ Les identifiants (noms de variables, de fonctions) doivent respecter les règles suivantes :
  - Contenir uniquement lettres Unicode, Chiffres, \$ et \_
  - Ne commencent pas par un chiffre
- ▶ Bonnes pratiques :
  - ne pas utiliser d'accents (passage d'un éditeur à un autre)
  - séparer les mots dans l'identifiant par des majuscules (camelCase), ou des \_ (snake\_case)
  - les identifiants qui commencent ou se terminent par des \$ ou \_ sont utilisées par certaines conventions
- ▶ Exemples :
  - Valides
    - i, maVariable, \$div, v1, prénom*
  - Invalides
    - ~~1var, ma-variable~~

# JavaScript - Mots clés



- ▶ Mots clés (ES10) :  
*await, break, case, catch, class, const, continue, debugger, default, delete, do, else, export, extends, finally, for, function, if, import, in, instanceof, new, return, super, switch, this, throw, try, typeof, var, void, while, with, yield*
- ▶ Mots clés (mode strict) :  
*let, static*
- ▶ Réservés pour une utilisation future :  
*enum*
- ▶ Réservés pour une utilisation future (mode strict) :  
*implements, interface, package, private, protected, public*



# JavaScript - Types

- ▶ Voici les types primitifs en JS
  - number
  - boolean
  - string
- ▶ Les types complexes
  - object
  - array
- ▶ Les types spéciaux
  - undefined
  - null

# JavaScript - Types



- ▶ Différence primitifs / complexes

En cas d'affectation ou de passage de paramètres, les primitifs ne sont pas modifiés, contrairement aux complexes

```
var boolean = false;
var number = 0;
var string = '';
var object = {};
var array = [];

var modify = function(b, n, s, o, a) {
    b = true;
    n = 1;
    s.concat('Romain'); // immuable
    o.prenom = 'Romain'; // object sera modifié également
    a.push('Romain'); // array sera modifié également
};

modify(boolean, number, string, object, array);

console.log(boolean); // false
console.log(number); // 0
console.log(string); // ''
console.log(object); // { prenom: 'Romain' }
console.log(array); // [ 'Romain' ]
```

# JavaScript - Number



- Pas de type spécifique pour les entiers ou les non-signés
- Implémentés en 64 bits en précision double
- Infinity et NaN sont 2 valeurs particulières de type number

```
// decimal
console.log(11); // 11
console.log(11.11); // 11.11

// binary
console.log(0b11); // 3 (ES6)

// octal
console.log(011); // 9
console.log(0o11); // 9 (ES6)

// hexadecimal
console.log(0x11); // 17

// exponentiation
console.log(1e3); // 1000

console.log(typeof 0); // 'number'
```

# JavaScript - NaN



- NaN est une valeur de type number pour les opérations impossibles (conversions, nombres complexes...)
- Une comparaison avec NaN donne systématiquement false (y compris NaN === NaN)

```
console.log(NaN); // NaN
console.log(Math.sqrt(-1)); // NaN
console.log(Number('abc')); // NaN
console.log(Number(undefined)); // NaN

console.log(typeof Math.sqrt(-1)); // number

console.log(NaN == NaN); // false
console.log(NaN === NaN); // false

console.log(isNaN(Math.sqrt(-1))); // true
console.log(Number.isNaN(Math.sqrt(-1))); // true (ES6)

console.log(isFinite(Math.sqrt(-1))); // false
console.log(Number.isFinite(Math.sqrt(-1))); // false (ES6)

console.log(0 < NaN); // false
console.log(0 > NaN); // false
console.log(0 == NaN); // false
console.log(0 === NaN); // false
```

# JavaScript - Infinity



- › Infinity est une valeur de type number, une division par zéro est donc possible en JS

```
console.log(Infinity); // Infinity
console.log(1 / 0); // Infinity

console.log(typeof (1 / 0)); // number

console.log(isFinite(1 / 0)); // false
console.log(Number.isFinite(1 / 0)); // false (ES6)

console.log(isNaN(1 / 0)); // false
console.log(Number.isNaN(1 / 0)); // false (ES6)

console.log(0 < Infinity); // true
console.log(0 > Infinity); // false
console.log(0 == Infinity); // false
console.log(0 === Infinity); // false
```

# JavaScript - Déclaration de variable



- ▶ Mot clé var

Contrairement à certains langages, on ne déclare pas le type au moment de la création (pas de typage statique)

```
var firstName = 'Romain';
var lastName = 'Bohdanowicz';
```

- ▶ Déclaration sans var

En cas de déclaration sans le mot clé var, la variable devient globale. Le mode strict apparu en ECMAScript 5 empêche ce comportement.

- ▶ ECMAScript 6

En ES6 une variable peut également se déclarer avec le mot clé let (portée de block), ou const (constante)



# JavaScript - Undefined

- ▶ Un identifiant qui n'est pas déclaré est typé undefined

```
var firstName;

console.log(firstName === undefined); // true
console.log(typeof firstName); // 'undefined'

console.log(lastName === undefined); // ReferenceError: lastName is not defined
console.log(typeof lastName); // 'undefined'
```

# JavaScript - Null



- › On utilise généralement null pour déréférencer un objet et ainsi permettre au garbage collector de libérer la mémoire associé
- › Dans certaines API, null est souvent utilisé en valeur de retour lorsqu'un objet est attendu mais qu'aucun objet ne convient.

```
var contacts = [{  
    firstName: 'Romain'  
}, {  
    firstName: 'Steven'  
}];  
  
function findContact(firstName, contacts) {  
    for (var i=0; i<contacts.length; i++) {  
        if (contacts[i].firstName === firstName) {  
            return contacts[i];  
        }  
    }  
  
    return null;  
}  
  
console.log(findContact('Jean', contacts)); // null;  
  
contacts = null; // dereference
```



# JavaScript - Opérateurs

## ▶ Affectation

Nom	Opérateur composé	Signification
Affectation	<code>x = y</code>	<code>x = y</code>
Affectation après addition	<code>x += y</code>	<code>x = x + y</code>
Affectation après soustraction	<code>x -= y</code>	<code>x = x - y</code>
Affectation après multiplication	<code>x *= y</code>	<code>x = x * y</code>
Affectation après division	<code>x /= y</code>	<code>x = x / y</code>
Affectation du reste	<code>x %= y</code>	<code>x = x % y</code>
Affectation après exponentiation	<code>x **= y</code>	<code>x = x ** y</code>

# JavaScript - Opérateurs



## ▶ Comparaison

Opérateur	Description	Exemples qui renvoient true
Égalité (==)	Renvoie true si les opérandes sont égaux après conversion en valeurs de mêmes types.	<code>3 == var1</code> <code>"3" == var1</code> <code>3 == '3'</code>
Inégalité (!=)	Renvoie true si les opérandes sont différents.	<code>var1 != 4</code> <code>var2 != "3"</code>
Égalité stricte (===)	Renvoie true si les opérandes sont égaux et de même type. Voir <code>Object.is()</code> et égalité de type en JavaScript.	<code>3 === var1</code>
Inégalité stricte (!==)	Renvoie true si les opérandes ne sont pas égaux ou s'ils ne sont pas de même type.	<code>var1 !== "3"</code> <code>3 !== '3'</code>
Supériorité stricte (>)	Renvoie true si l'opérande gauche est supérieur (strictement) à l'opérande droit.	<code>var2 &gt; var1</code> <code>"12" &gt; 2</code>
Supériorité ou égalité (>=)	Renvoie true si l'opérande gauche est supérieur ou égal à l'opérande droit.	<code>var2 &gt;= var1</code> <code>var1 &gt;= 3</code>
Infériorité stricte (<)	Renvoie true si l'opérande gauche est inférieur (strictement) à l'opérande droit.	<code>var1 &lt; var2</code> <code>"2" &lt; "12"</code>
Infériorité ou égalité (<=)	Renvoie true si l'opérande gauche est inférieur ou égal à l'opérande droit.	<code>var1 &lt;= var2</code> <code>var2 &lt;= 5</code>

# JavaScript - Opérateurs



## ► Arithmétiques

En plus des opérations arithmétiques standards (+, -, \*, /), on trouve en JS :

Opérateur	Description	Exemple
Reste (%)	Opérateur binaire. Renvoie le reste entier de la division entre les deux opérandes.	12 % 5 renvoie 2.
Incrément (++)	Opérateur unaire. Ajoute un à son opérande. S'il est utilisé en préfixe (++x), il renvoie la valeur de l'opérande après avoir ajouté un, s'il est utilisé comme opérateur de suffixe (x++), il renvoie la valeur de l'opérande avant d'ajouter un.	Si x vaut 3, ++x incrémente x à 4 et renvoie 4, x++ renvoie 3 et seulement ensuite ajoute un à x.
Décrément (--)	Opérateur unaire. Il soustrait un à son opérande. Il fonctionne de manière analogue à l'opérateur d'incrément.	Si x vaut 3, --x décrémente x à 2 puis renvoie 2, x-- renvoie 3 puis décrémente la valeur de x.
Négation unaire (-)	Opérateur unaire. Renvoie la valeur opposée de l'opérande.	Si x vaut 3, alors -x renvoie -3.
Plus unaire (+)	Opérateur unaire. Si l'opérande n'est pas un nombre, il tente de le convertir en une valeur numérique.	+ "3" renvoie 3. + true renvoie 1.
Opérateur d'exponentiation (**)(puissance)	Calcule un nombre (base) élevé à une puissance donnée (soit basepuissance) (ES7)	2 ** 3 renvoie 8 10 ** -1 renvoie -1

# JavaScript - Opérateurs



## ▶ Logiques

Opérateur	Usage	Description
ET logique (&&)	expr1 && expr2	Renvoie expr1 s'il peut être converti à false, sinon renvoie expr2. Dans le cas où on utilise des opérandes booléens, && renvoie true si les deux opérandes valent true, false sinon.
OU logique (  )	expr1    expr2	Renvoie expr1 s'il peut être converti à true, sinon renvoie expr2. Dans le cas où on utilise des opérandes booléens,    renvoie true si l'un des opérandes vaut true, si les deux valent false, il renvoie false.
NON logique (!)	!expr	Renvoie false si son unique opérande peut être converti en true, sinon il renvoie true.

# JavaScript - Opérateurs



- ▶ Concaténation

```
console.log('ma ' + 'chaîne'); // affichera "ma chaîne" dans la console
```

- ▶ Ternaire

```
var statut = (âge >= 18) ? 'adulte' : 'mineur';
```

- ▶ Voir aussi
  - Opérateurs binaires, in, instanceof, delete, typeof...
- ▶ Attention au '+' qui donne priorité à la concaténation

```
console.log('1' + '1' + '1'); // '111'  
console.log('1' + '1' + 1); // '111'  
console.log('1' + 1 + 1); // '111'  
console.log( 1 + 1 + '1'); // '21'
```

# JavaScript - Opérateurs



## ▶ Priorités

Type d'opérateur	Opérateurs individuels
membre	. []
appel/création d'instance	() new
négation/incrémantion	! ~ - + ++ -- typeof void delete
multiplication/division	* / %
addition/soustraction	+ -
décalage binaire	<< >> >>>
relationnel	< <= > >= in instanceof
égalité	== != === !==
ET binaire	&
OU exclusif binaire	^
OU binaire	
ET logique	&&
OU logique	
conditionnel	? :
assignation	= += -= *= /= %= <<= >>= >>>= &= ^=  =
virgule	,



# JavaScript - Conversions

- ▶ Conversions implicites

```
console.log(3 * '3'); // 9  
console.log(3 + '3'); // '33'  
console.log(!'texte'); // false
```

- ▶ Conversions explicites

```
console.log(parseInt('33.33')); // 33  
console.log(parseFloat('33.33')); // 33.33  
console.log(Number('33.33')); // 33.33  
console.log(Boolean('texte')); // true  
console.log(String(33.33)); // '33.33'
```

- ▶ Les conversions de types

<https://www.youtube.com/watch?v=cueiAe7JlfY>

# JavaScript - API



- ▶ Standard Built-in Objects

Les objets prédéfinies par le langage, voir la doc de Mozilla pour une liste exhaustive

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/  
Global\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects)

- ▶ Ex : String, Array, Date, Math, RegExp, JSON...

# JavaScript - Tableaux



## ▶ Structure et API

En JS les tableaux ne sont pas des structures de données mais un type d'objet (une « classe »).

```
var firstNames = ['Romain', 'Eric'];

console.log(firstNames.length); // 2

console.log(firstNames[0]); // Romain
console.log(firstNames[firstNames.length - 1]); // Eric

// boucler sur tous les éléments (ES5)
firstNames.forEach(function(firstName) {
  console.log(firstName); // Romain Eric
});

var newLength = firstNames.push('Jean'); // ajoute Jean à la fin
var last = firstNames.pop(); // retire et retourne le dernier (Jean)
var newLength = firstNames.unshift("Jean") // ajoute Jean au début
var first = firstNames.shift(); // retire et retourne le premier (Jean)

var pos = firstNames.indexOf("Romain"); // indice de l'élément
var removedItem = firstNames.splice(pos, 1); // suppression d'un élément à
partir de l'indice pos
var shallowCopy = firstNames.slice(); // copie d'un tableau
```

# JavaScript - Structures de contrôle



- ▶ if ... else

```
if (typeof console === 'object') {  
    console.log('console est un objet');  
}  
else {  
    // oups  
}
```

- ▶ switch

```
switch (alea) {  
    case 0:  
        console.log('zéro');  
        break;  
    case 1:  
    case 2:  
    case 3:  
        console.log('un, deux ou trois');  
        break;  
    default:  
        console.log('entre quatre et neuf');  
}
```



# JavaScript - Structures de contrôle

- while

```
var alea = Math.floor(Math.random() * 10);

while (alea > 0) {
    console.log(alea);
    alea = parseInt(alea / 2);
}
```

- do ... while

```
do {
    var alea = Math.floor(Math.random() * 10);
}
while (alea % 2 === 1);

console.log(alea);
```

- for

```
for (var i=0; i<10; i++) {
    aleas.push(Math.floor(Math.random() * 10));
}

console.log(aleas.join(', ')); // 6, 6, 7, 0, 5, 1, 2, 8, 9, 7
```



**formation.tech**

# Fonctions en JavaScript (ECMAScript 3)

# Fonctions en JavaScript - Introduction



- ▶ JavaScript est très consommateur de fonctions
  - réutilisation / factorisation
  - récursivité
  - fonction de rappel (callback) / écouteur (listener)
  - closure
  - module



# Fonctions en JavaScript - Syntaxe

- ▶ Function declaration

```
function addition(nb1, nb2) {  
    return Number(nb1) + Number(nb2);  
}  
  
console.log(addition(2, 3)); // 5
```

- ▶ Anonymous function expression

```
var addition = function (nb1, nb2) {  
    return Number(nb1) + Number(nb2);  
}  
  
console.log(addition(2, 3)); // 5
```

- ▶ Named function expression

```
var addition = function addition(nb1, nb2) {  
    return Number(nb1) + Number(nb2);  
}  
  
console.log(addition(2, 3)); // 5
```

# Fonctions en JavaScript - Function Declaration



- › En JavaScript, les fonctions et variables sont hissées (hoisted) au début de la portée dans laquelle elles ont été déclarée.
- › Il est donc possible d'appeler une fonction avant sa déclaration
- › Pas d'erreur en cas de redéclaration de fonctions, la seconde écrase la première

```
function hello() {  
    return 'Hello 1';  
}  
  
console.log(hello()); // 'Hello 2'  
  
function hello() {  
    return 'Hello 2';  
}
```

# Fonctions en JavaScript - Function Expression



- Avec une function expression, la variable est hissée en début de portée
- Mais la fonction est créée au moment où l'expression s'exécute

```
var hello = function () {
  return 'Hello 1';
};

console.log(hello()); // 'Hello 1'

var hello = function () {
  return 'Hello 2';
};
```



# Fonctions en JavaScript - Constantes

- En ES6 on pourrait même empêcher la redéclaration grâce au mot clé const

```
const hello = function () {
  return 'Hello 1';
};

console.log(hello());

// SyntaxError: Identifier 'hello' has already been declared
const hello = function () {
  return 'Hello 2';
};
```

# Fonctions en JavaScript - Named Function Expression



- Anonymous function expression vs Named function expression

```
document.addEventListener('click', function() {
  ['Romain', 'Eric'].forEach(function(firstName) {
    console.log(firstName);
  });
});
```

The screenshot shows a browser developer tools debugger interface. The code being debugged is:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8 <script>
9
10  document.addEventListener('click', function() {
11    ['Romain', 'Eric'].forEach(function(firstName) {
12      console.log(firstName);
13    });
14  });
15 </script>
```

The line `12 console.log(firstName);` is highlighted with a blue selection bar. The debugger sidebar shows:

- Call Stack:
  - (anonymous function)
  - (anonymous function)
- Scope
  - Local
    - firstName: "Romain"
    - this: Window
  - Global

Message: Paused on a JavaScript breakpoint.

```
document.addEventListener('click', function clickHandler() {
  ['Romain', 'Eric'].forEach(function forEachFirstName(firstName) {
    console.log(firstName);
  });
});
```

The screenshot shows a browser developer tools debugger interface. The code being debugged is:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8 <script>
9
10  document.addEventListener('click', function clickHandler() {
11    ['Romain', 'Eric'].forEach(function forEachFirstName(firstName) {
12      console.log(firstName);
13    });
14  });
15 </script>
```

The line `12 console.log(firstName);` is highlighted with a blue selection bar. The debugger sidebar shows:

- Call Stack:
  - forEachFirstName
  - clickHandler
- Scope
  - Local
    - firstName: "Romain"
    - this: Window
  - Global

Message: Paused on a JavaScript breakpoint.



# Fonctions en JavaScript - Paramètres

## ▶ Paramètres

Comme pour les variables, on ne déclare pas les types des paramètres d'entrées et de retours.

Les paramètres ne font pas partie de la signature de la fonction, seul l'identifiant compte, on peut donc appeler une fonction avec plus ou moins de paramètres que prévu.

```
var sum = function(a, b) {  
    return a + b;  
};  
  
console.log(sum(1, 2)); // 3  
console.log(sum('1', '2')); // '12'  
console.log(sum(1, 2, 3)); // 3  
console.log(sum(1)); // NaN
```

# Fonctions en JavaScript - Exceptions



- ▶ Exceptions
  - En cas d'utilisation anormale d'une fonction, on peut sortir en lançant une exception.
- ▶ N'importe quel type peut être envoyé via le mot clé `throw`, mais privilégier les objets de type `Error` et dérivés qui interceptent les fichiers, pile d'appel et numéro de lignes.
- ▶ On ne peut pas intercepter une exception avec `try..catch` si elle est lancée dans un callback asynchrone

```
var sum = function(a, b) {  
    if (typeof a !== 'number' || typeof b !== 'number') {  
        throw new Error('sum needs 2 number')  
    }  
    return a + b;  
};  
  
try {  
    sum('1', '2'); // sum needs 2 number  
}  
catch (err) {  
    console.log(err.message);  
}
```

# Fonctions en JavaScript - Valeur par défaut



## ▶ Valeur par défaut

Les paramètres non renseignées lors de l'appel d'une fonction reçoivent la valeur `undefined`.

```
// using undefined
var sum = function(a, b, c) {
  if (c === undefined) {
    c = 0;
  }
  return a + b + c;
};

console.log(sum(1, 2)); // 3

// using or (si la valeur par défaut est falsy uniquement)
var sum = function(a, b, c) {
  c = c || 0;
  return a + b + c;
};

console.log(sum(1, 2)); // 3
```

# Fonctions en JavaScript - Paramètres non déclarés



## ▶ Fonction Variadique

Pour récupérer les paramètres supplémentaires (non déclarés), on peut utiliser la variable `arguments`. Cette variable n'étant pas un tableau, on ne peut pas utiliser les fonctions du type `Array` (même si des astuces existent).

```
var sum = function(a, b) {
    var result = a + b;

    for (var i=2; i<arguments.length; i++) {
        result += arguments[i];
    }

    return result;
};

console.log(sum(1, 2, 3, 4)); // 10
```



# Fonctions en JavaScript - Imbrication

- ▶ Fonctions imbriquées

En JavaScript on peut imbriquer les fonctions, la portée d'une fonction étant la fonction qui la contient.

```
var sumArray = function(array) {  
    var sum = function(a, b) {  
        return a + b;  
    };  
    return array.reduce(sum);  
};  
  
console.log(sumArray([1, 2, 3, 4])); // 10  
console.log(typeof sum); // 'undefined'
```

# Fonctions en JavaScript - Portées



- ▶ Portées

Lorsque l'on imbrique des fonctions, les portées supérieures restent accessibles.

```
var a = function() {
  var b = function() {
    var c = function() {
      console.log(typeof a); // function
      console.log(typeof b); // function
      console.log(typeof c); // function
    };
    c();
  };
  b();
};
a();
```

- ▶ Pas besoin de repasser les variables en paramètres si les fonctions sont imbriquées



# Fonctions en JavaScript - Closure

## ▶ Closure

Si 2 fonctions sont imbriquées et que la fonction interne est appelée en dehors (par valeur de retour ou asynchronisme), on parle de closure.

La portée des variables au moment du passage dans la fonction externe est sauvegardée.

The screenshot shows a browser developer tools debugger interface with a file named "closure.js" open. The code is as follows:

```
var logClosure = function(msg) {
    return function() {
        console.log(msg);
    };
};

var logHello = logClosure('Hello')
logHello(); // Hello
```

The debugger highlights the line `console.log(msg);` at line 3. The call stack shows the execution path: an anonymous function at line 3, followed by another anonymous function at line 8, which is the `logHello` function. The scope section shows the local variable `this` pointing to `Window`, and a closure variable `msg` with the value `"Hello"`. The global scope includes standard browser objects like `Infinity`, `AnalyserNode`, and `AnimationEvent`.



# Fonctions en JavaScript - Exemple de Closure

- ▶ Sans Closure

```
// affiche 4 4 4 dans 1 seconde
for (var i = 1; i <= 3; i++) {
    setTimeout(function() {
        console.log(i);
    }, 1000);
}
```

- ▶ Avec Closure

```
// affiche 1 2 3 dans 1 seconde
for (var i = 1; i <= 3; i++) {
    setTimeout(function(rememberI) {
        return function() {
            console.log(rememberI);
        };
    }(i), 1000);
}
```

# Fonctions en JavaScript - Callbacks



- ▶ **Callback**

Lorsqu'un fonction est passée en paramètre d'entrée d'une autre fonction en vue d'être appelée plus tard, on parle de callback.

- ▶ **Callback synchrone / asynchrone**

Une fonction recevant un callback peut être synchrone, c'est à dire qu'elle doit s'exécuter entièrement avant d'appeler les instructions suivantes, ou asynchrone ce qui signifie que la fonction sera appelée dans un prochain passage de la « boucle d'événements »

```
var firstNames = ['Romain', 'Eric'];

firstNames.forEach(function(firstName) {
  console.log(firstName);
});

setTimeout(function() {
  console.log('Hello in 100ms');
}, 100);
```

# Fonctions en JavaScript - Callback Synchrone



- API recevant un callback synchrone

```
var firstNames = ['Romain', 'Eric'];

var forEachSync = function(array, callback) {
    for (var i=0; i<array.length; i++) {
        callback(array[i], i, array);
    }
};

forEachSync(firstNames, function(firstName) {
    console.log(firstName);
});

console.log('After forEachSync');

// Outputs :
// Romain
// Eric
// After forEachSync
```

# Fonctions en JavaScript - Callback Asynchrone



- API recevant un callback asynchrone

```
var firstNames = ['Romain', 'Eric'];

var forEachASync = function(array, callback) {
    for (var i=0; i<array.length; i++) {
        setTimeout(callback, 0, array[i], i, array);
    }
};

forEachASync(firstNames, function(firstName) {
    console.log(firstName);
});

console.log('After forEachASync');

// Outputs :
// After forEachASync
// Romain
// Eric
```

# Fonctions en JavaScript - Boucle d'événements



- › Les moteurs JS sont par défaut mono-thread et mono-processus, ils ne peuvent donc exécuter qu'une seule tâche à la fois.
- › Une boucle d'événements permet de passer d'un callback à l'autre de manière très performante, ex : traiter le clic d'un bouton entre 2 étapes d'une animation
- › JavaScript est non-bloquant, il stocke les événements à traiter sous la forme d'une file de message et appellera les callbacks lorsqu'il sera disponible
- › Bonne pratique : les callbacks doivent avoir un temps d'exécution court pour ne pas ralentir l'appel des callbacks suivants

```
setTimeout(function() {
    console.log('1 fois dans 3 secondes');
}, 3000);

var intervalId = setInterval(function() {
    console.log('toutes les 2 secondes');
}, 2000);

setTimeout(function() {
    console.log('Bye bye');
    clearInterval(intervalId);
}, 15000);
```

# Fonctions en JavaScript - Boucle d'événements



- ▶ Boucle d'événements

Lorsqu'un programme JS est démarré, il tourne dans une boucle d'événements.

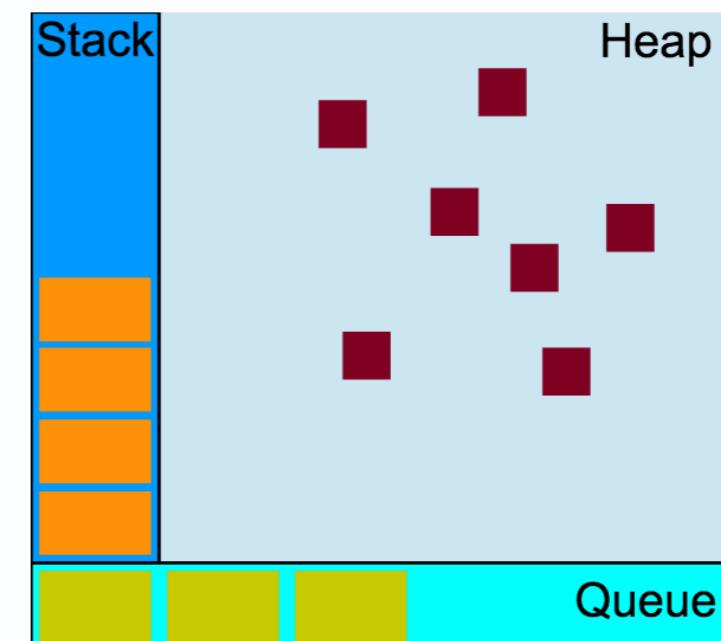
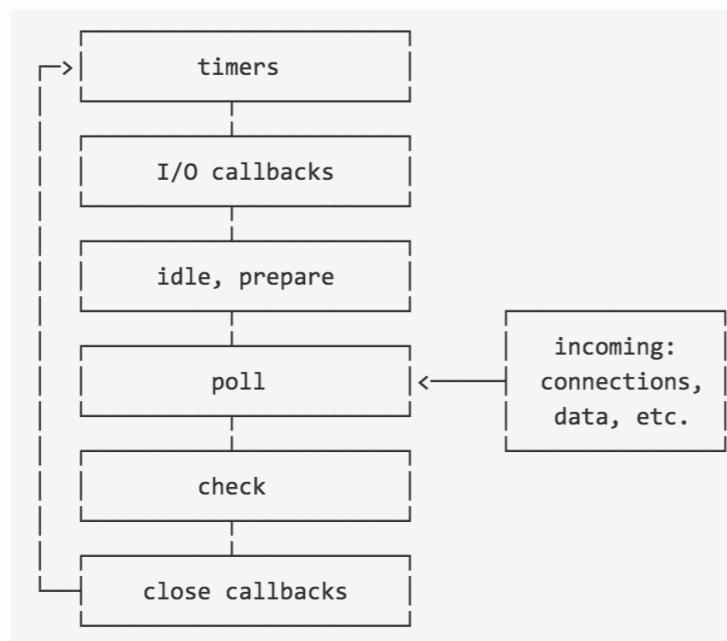
Tant qu'il y a des appels en cours dans la pile d'appels, où des callbacks en attente dans la file de callback, on ne passe pas à la prochain itération. Dans le navigateur, un seul thread est en charge du JavaScript et du rendu, pour un rendu à 60FPS il faut qu'un passage dans la boucle JS + rendu ne dépasse pas 16,67ms.

- ▶ What the heck is the event loop anyway? | JSConf EU 2014

<https://www.youtube.com/watch?v=8aGhZQkoFbQ>

- ▶ Jake Archibald: In The Loop - JSConf.Asia 2018

<https://www.youtube.com/watch?v=cCOL7MC4Pl0>

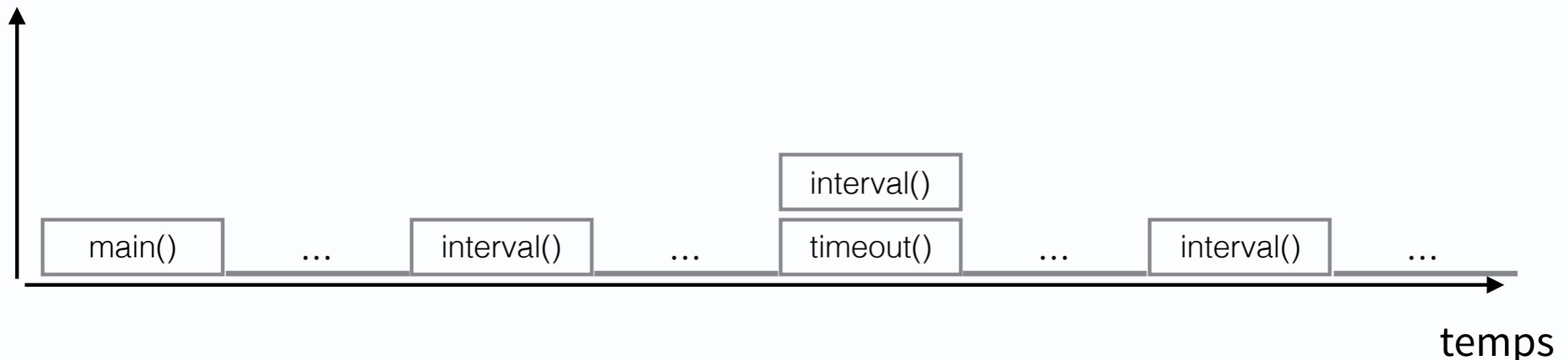




# Fonctions en JavaScript - Boucle d'événements

- ▶ Boucle d'événements

File d'attente



```
setInterval(function interval() {  
    console.log('interval 1ms')  
}, 1000);  
  
setTimeout(function timeout() {  
    console.log('timeout 2ms')  
}, 2000);
```



# Fonctions en JavaScript - API Function

- ▶ Object function

```
var contact = {  
    prenom: 'Romain',  
    nom: 'Bohdanowicz'  
};  
  
function saluer(prenom) {  
    return 'Bonjour ' + prenom + ' je suis ' + this.prenom;  
}  
  
console.log(saluer('Eric')); // Bonjour Eric je suis undefined  
console.log(saluer.call(contact, 'Eric')); // Bonjour Eric je suis Romain  
console.log(saluer.apply(contact, ['Eric'])); // Bonjour Eric je suis Romain
```



# Fonctions en JavaScript - Modules

## ▶ Module

Contrairement à Node.js, il n'y a pas de portée de fichier dans le navigateur, pour éviter les conflits de nom, on utilise généralement des fonctions anonymes pour créer une portée de fichier, c'est la notion de Module.

## ▶ Immediately Invoked Function Expression (IIFE)

```
(function($, global) {
    'use strict';

    function MonBouton(options) {
        this.options = options || {};
        this.value = options.value || 'Valider';
    }

    MonBouton.prototype.creer = function(container) {
        $(container).append('<button>' + this.value + '</button>');
    };

    global.MonBouton = MonBouton;
})(jQuery, window);
```

# Fonctions en JavaScript - Exercice



- Jeu du plus ou moins
  - 1.Générer un entier aléatoire entre 0 et 100 (API Math sur MDN)
  - 2.Demander et récupérer la saisie, afficher si le nombre est plus grand, plus petit ou trouvé (API Readline sur Node.js)
  - 3.Pouvoir trouver en plusieurs tentatives (problème d'asynchronisme)
  - 4.Stocker les essais dans un tableau et les réafficher entre chaque tour (API Array sur MDN)
  - 5.Afficher une erreur si la saisie n'est pas un nombre (API Number sur MDN)
- Attention, le callback de question est toujours appelé avec un type String, à convertir si besoin.



**formation.tech**

# JavaScript Orienté Objet (ECMAScript 3)

# JavaScript Orienté Objet - Introduction



- JavaScript a un modèle objet orienté prototype
- Modèle statique vs Modèle dynamique
  - Il n'y a pas de définition statique du type ou du contenu d'un objet, l'ajout de propriété ou de méthode se fait dynamiquement
- Classe vs dictionnaires
  - La notion de classe ou d'interface n'existe pas (seulement dans les docs où sous la forme de sucre syntaxique) à la place l'objet JavaScript est un dictionnaire tel que :
    - PHP : tableaux associatifs
    - Java, C++ : Map, HashTable
    - C : Struct
    - C# : Dictionary
    - Python : dictionary
    - Ruby : Hash

# JavaScript Orienté Objet - Objets préinstanciés



- ▶ Il y a un certain nombre d'objet définis au niveau du langage

```
Math.random();
JSON.stringify({});
console.log(typeof Math); // object
console.log(typeof JSON); // object
```

- ▶ D'autres par l'environnement d'exécution (Node.js, Navigateur, Mobile...)

```
console.log(typeof console); // object (dans le navigateur et Node.js)
console.log(typeof document); // object (dans le navigateur)
```

# JavaScript Orienté Objet - Extensibilité



## ▶ Extensibilité

On peut étendre (sauf verrou), n'importe quel objet. Etendre les objets standards est cependant considéré comme une mauvaise pratique (sauf polyfill). Attention à la casse lorsque vous modifiez une propriété.

```
Math.sum = function(a, b) {  
    return a + b;  
};  
console.log(Math.sum(1, 2)); // 3
```

## ▶ On peut également modifier ou supprimer des propriétés

```
var randomBackup = Math.random;  
Math.random = function() {  
    return 0.5;  
};  
  
console.log(Math.random()); // 0.5  
Math.random = randomBackup;  
console.log(Math.random()); // quelque chose aléatoire comme 0.24554522  
  
delete Math.sum;  
console.log(Math.sum); // undefined
```

# JavaScript Orienté Objet - Objets ponctuels



- Création d'un objet avec l'objet global Object (mauvaise pratique) :

```
var instructor = new Object();
instructor.firstName = 'Romain';
instructor.hello = function() {
    return 'Hello my name is ' + this.firstName;
};

console.log(instructor.hello()); // Hello my name is Romain
```

- Création d'un objet avec la syntaxe Object Literal (recommandé) :

```
var instructor = {
    firstName: 'Romain',
    hello: function() {
        return 'Hello my name is ' + this.firstName;
    }
};

console.log(instructor.hello()); // Hello my name is Romain
```



# JavaScript Orienté Objet - Opérateurs

- ▶ Accès aux objets possible :
  - Avec l'opérateur `.`
  - Avec des crochets

```
var instructor = {
  firstName: 'Romain',
  hello: function() {
    return 'Hello my name is ' + this.firstName;
  }
};

instructor.firstName = 'Jean';
console.log(instructor.hello()); // Hello my name is Jean

instructor['firstName'] = 'Eric';
console.log(instructor['hello']()); // Hello my name is Eric
```

# JavaScript Orienté Objet - Fonction Constructeur



- En utilisant une fonction constructeur (avec closure, mauvaise pratique) :

```
var Person = function (firstName) {
    this.firstName = firstName;

    this.hello = function () {
        // firstName existe aussi grâce à la closure
        return 'Hello my name is ' + this.firstName;
    };
};

var instructor = new Person('Romain');

console.log(instructor.hello()); // Hello my name is Romain
console.log(typeof instructor); // object
console.log(instructor instanceof Object); // true
console.log(instructor instanceof Person); // true

for (var prop in instructor) {
    if (instructor.hasOwnProperty(prop)) {
        console.log(prop); // firstName puis hello
    }
}
```



# JavaScript Orienté Objet - Fonction Constructeur

- En utilisant une fonction constructeur + son prototype :

```
var Person = function(firstName) {
    this.firstName = firstName;
};

Person.prototype.hello = function () {
    return 'Hello my name is ' + this.firstName;
};

var instructor = new Person('Romain');

console.log(instructor.hello()); // Hello my name is Romain
console.log(typeof instructor); // object
console.log(instructor instanceof Object); // true
console.log(instructor instanceof Person); // true

for (var prop in instructor) {
    if (instructor.hasOwnProperty(prop)) {
        console.log(prop); // firstName
    }
}
```

# JavaScript Orienté Objet - Héritage



- En utilisant une fonction constructeur + son prototype :

```
var Instructor = function(firstName, speciality) {
    Person.apply(this, arguments); // héritage des propriétés de l'objet (recopie dynamique)
    this.speciality = speciality;
}

Instructor.prototype = new Person; // héritage du type

// Redéfinition de méthode
Instructor.prototype.hello = function() {
    // Appel de la méthode parent
    return Person.prototype.hello.call(this) + ', my speciality is ' + this.speciality;
};

var instructor = new Instructor('Romain', 'JavaScript');

console.log(instructor.hello()); // Hello my name is Romain
console.log(typeof instructor); // object
console.log(instructor instanceof Object); // true
console.log(instructor instanceof Person); // true
console.log(instructor instanceof Instructor); // true

for (var prop in instructor) {
    if (instructor.hasOwnProperty(prop)) {
        console.log(prop); // firstName, speciality
    }
}
```

# JavaScript Orienté Objet - Prototype



- ▶ Définition Wikipedia :

La programmation orientée prototype est une forme de programmation orientée objet sans classe, basée sur la notion de prototype. Un prototype est un objet à partir duquel on crée de nouveaux objets.

- ▶ Comparaison des modèles à classes et à prototypes

- Objets à classes :

- Une classe définie par son code source est statique ;
    - Elle représente une définition abstraite de l'objet ;
    - Tout objet est instance d'une classe ;
    - L'héritage se situe au niveau des classes.

- Objets à prototypes :

- Un prototype défini par son code source est mutable ;
    - Il est lui-même un objet au même titre que les autres ;
    - Il a donc une existence physique en mémoire ;
    - Il peut être modifié, appelé ;
    - Il est obligatoirement nommé ;
    - Un prototype peut être vu comme un exemplaire modèle d'une famille d'objet ;
    - Un objet hérite des propriétés (valeurs et méthodes) de son prototype ;

# JavaScript Orienté Objet - Prototype



- ▶ En ECMAScript/JavaScript, l'écriture `foo.bar` s'interprète de la façon suivante :
  1. Le nom `foo` est recherché dans la liste des identificateurs déclarés dans le contexte d'appel de fonction courant (déclarés par `var`, ou comme paramètre de la fonction) ;
  2. S'il n'est pas trouvé :
    - Continuer la recherche (retour à l'étape 1) dans le contexte de niveau supérieur (s'il existe),
    - Sinon, le contexte global est atteint, et la recherche se termine par une erreur de référence.
  3. Si la valeur associée à `foo` n'est pas un objet, il n'a pas de propriétés ; la recherche se termine par une erreur de référence.
  4. La propriété `bar` est d'abord recherchée dans l'objet lui-même ;
  5. Si la propriété ne s'y trouve pas :
    - Continuer la recherche (retour à l'étape 4) dans le prototype de cet objet (s'il existe) ;
    - Si l'objet n'a pas de prototype associé, la valeur indéfinie (`undefined`) est retournée ;
  6. Sinon, la propriété a été trouvée et sa référence est retournée.

# JavaScript Orienté Objet - JSON



- JSON, JavaScript Object Notation est la sérialisation d'un objet JavaScript
- Seuls les types string, number, boolean, array, object, regexp sont sérialisable, les fonctions et prototype sont perdus
- On se sert de ce format pour échanger des données entre 2 programmes ou pour créer de la config
- Le format résultant est proche de Object Literal, les clés sont obligatoirement entre guillemets "", un code JSON est une syntaxe Object Literal valide

```
{  
  "name": "My Address Book",  
  "contacts": [  
    {  
      "firstName": "Bill",  
      "lastName": "Gates"  
    },  
    {  
      "firstName": "Steve",  
      "lastName": "Jobs"  
    }  
  ]  
}
```

# JavaScript Orienté Objet - JSON vs XML



- › JSON est souvent utilisé en remplaçant d'XML pour des fichiers de configuration ou des échanges réseaux (entre un client et un serveur par exemple), car plus lisible, moins verbeux et plus simple à manipuler dans le code
- › L'exemple représentant la même structure de données, contient hors espaces et retours à la ligne : 115 caractères en JSON contre 217 en XML

```
{  
  "name": "Address Book",  
  "contacts": [  
    { "firstName": "Bill", "lastName": "Gates" },  
    { "firstName": "Steve", "lastName": "Jobs" }  
  ]  
}
```

```
<addressBook>  
  <name>My Adress Book</name>  
  <contacts>  
    <contact>  
      <firstName>Bill</firstName>  
      <lastName>Gates</lastName>  
    </contact>  
    <contact>  
      <firstName>Steve</firstName>  
      <lastName>Jobs</lastName>  
    </contact>  
  </contacts>  
</addressBook>
```

# JavaScript Orienté Objet - JSON



- JavaScript depuis ECMAScript 5 fourni l'objet global JSON qui contient 2 méthodes, parse (déserialiser) et stringify (sérialiser)

```
var contact = {  
    prenom: 'Romain',  
    nom: 'Bohdanowicz'  
};  
  
var json = JSON.stringify(contact);  
console.log(json); // {"prenom":"Romain","nom":"Bohdanowicz"}  
  
var object = JSON.parse(json);  
console.log(object.prenom); // Romain
```

# JavaScript Orienté Objet - Exercice



- Reprendre le jeu du plus ou moins
- Créer un objet random avec la syntaxe Object Literal et y regrouper les fonctions aléatoires
- Créer une fonction constructeur Jeu recevant un objet en paramètres d'entrée
- Créer une méthode jouer() tel que le code suivant soit fonctionnel
- Prévoir des valeurs par défaut pour min et max

```
const random = {  
  ...  
};  
  
function Jeu(options) { ... };  
  
// ....  
  
const game = new Jeu({  
  min: 0,  
  max: 100  
});  
  
game.jouer();
```



**formation.tech**

# Expression Régulières (ECMAScript 3)

# Expression Régulières - Introduction



- Les expression régulières sont un moyen en informatique de traiter des formats de chaînes de caractères
- Elles permettent de :
  - valider des saisies utilisateur (email, code postal...)
  - extraire des valeurs

# Expression Régulières - Création



- ▶ Depuis la première version de JavaScript, un objet global RegExp permet de manipuler les expressions régulières
- ▶ On peut utiliser une fonction constructeur ou une syntaxe littérale

```
var regex1 = /\w+;  
var regex2 = new RegExp('\\\w+');
```

- ▶ Privilégier la syntaxe littérale qui est plus performante, la fonction constructeur elle est plus dynamique puisqu'elle créé la RegExp à partie d'une chaîne de caractère qui pourrait être issue d'une saisie utilisateur

# Expression Régulières - Format



- ▶ Une suite de caractères permet de matcher tout ou partie d'une chaîne :
  - /ab/ va matcher **ab**, **abc** ou baob**ab**
  - /main/ va matcher **main**, **Romain** ou **maintenant**
- ▶ Pour indiquer que l'on recherche au début ou à la fin (ou les deux) on commence par ^ ou on termine par \$ :
  - /^ba/ va matcher **bateau** mais pas cabas
  - /en\$/ va matcher **chien** mais pas pente
  - /^eau\$/ va matcher **eau** mais pas château ou poteaux
- ▶ Pour indiquer qu'un caractère peut faire partie d'une liste on utilise les métacaractères [] :
  - /^cha[tp]eau\$/ va matcher **chateau** et **chapeau**
  - /^ [abc] / va matcher **bateau**, **ananas**, **chat** mais pas **drapeau**



# Expression Régulières - Format

- On peut également indiquer une liste de caractères [a-f] est l'équivalent [abcdef] (les caractères doivent se suivrent dans le code ASCII) :
  - / [a-z]\$ / va matcher **arte** mais pas tf1
  - / ^ [a-fw-z] / va matcher **avion**, **wagon** mais pas **piéton**
  - / ^ [0-9] / va matcher **2019** mais pas **bateau**
- Les expressions régulières sont sensibles à la casse, il faudra donc inclure la majuscule et la minuscule dans le méta-caractère :
  - / ^ [Rr] / va matcher **Romain** et **romain**
- Les caractères accentués ne sont pas inclus dans la liste [a-z], il faudra utiliser leur code Unicode (<https://unicode-table.com/fr/>) :
  - / ^ [\u00c0-\u00ff] / va matcher **ça** ou **à**



# Expression Régulières - Format

- Si un méta-caractère commence par ^ cela signifie que le caractère ne doit pas faire partie de la liste
  - /[^aeiouy]\$/ va matcher vin mais pas eau
- On peut créer un groupe avec des parenthèses (permettra en JS de capturer ce groupe)
  - /( [1-9] [0-9] ) / var matcher **75** mais pas 06
- On peut proposer plusieurs alternatives avec le |
  - /bon(jour | soir) / va matcher **bonjour** et **bonsoir**
- Préfixer un groupe par ?: le rend non-capturant (en JS)
  - /(?: [1-9] [0-9] ) / var matcher **75** mais pas 06 (non capturant)



# Expression Régulières - Format

- ▶ Certains méta-caractères ont des raccourcis
  - \w est l'équivalent de [A-Za-z0-9\_]
  - \W est l'équivalent de [^A-Za-z0-9\_]
  - \d est l'équivalent de [0-9]
  - \D est l'équivalent de [^0-9]
  - \s est l'équivalent de [\t\r\n\v\f]
  - \S est l'équivalent de [^\t\r\n\v\f]
  - . est l'équivalent de n'importe quel caractère
- ▶ Il est souvent nécessaire d'échapper certains caractères
  - /\d.\d/ matche **2.3** mais aussi **2b3**
  - /\d\.\d/ matche **2.3** mais par 2b3



# Expression Régulières - Format

- ▶ Pour matcher plusieurs caractères, méta-caractères ou groupe on peut utiliser :
  - {n} doit apparaître n fois, `/^a{3}/` va matcher **aaaaaa**
  - {n,m} doit apparaître en n et m fois, `/^a{2,3}/` va matcher **aaaaaa** (match le plus long possible)
  - {n,} doit apparaître au moins n fois, `/^a{2,}/` va matcher **aaaaaa** (match le plus long possible)
  - ? signifie 0 ou 1 fois, `/^a?/` va matcher **aaaaaa**
  - \* signifie 0, 1 ou plusieurs fois, `/^a*/` va matcher **aaaaaa**
  - + signifie 1 ou plusieurs fois, `/^a+/` va matcher **aaaaaa**
- ▶ Exemples
  - `/[1-9][0-9]*/` un nombre entier positif (chiffre entre 1 et 9 suivi de 0, 1 ou plusieurs chiffres)
  - `/(pa){2}/` va matcher **papa**



# Expression Régulières - Format

- L'option (flag) g permet de répéter la recherche (par défaut, la regex s'arrête au premier match)
  - /a/g va matcher **aaa** puis **aaa** puis **aaa**
- L'option (flag) i pour case insensitive, recherche insensible à la casse
  - /[a-z]+/i va matcher **abc**, **ABC** ou **AbC**
- L'option (flag) m permet que les caractères ^ et \$ s'appliquent à la ligne et non l'ensemble de la chaîne de caractères
  - /[a-z]+/im va matcher  
**abcd**  
efgh



# Expression Régulières - regex.test

- › La méthode *test* retourne true lorsque la chaîne de caractère match l'expression régulière
- › La propriété *lastIndex* est incrémenté et correspond à la position du curseur à la fin du match (le flag g permet de repartir de cet index)

```
const regex = /\d+/g;
const str = '01/10/1985';

console.log(regex.test(str)); // true

regex.lastIndex = 0; // revient au début de str

while (regex.test(str)) {
  console.log(regex.lastIndex); // 2, 5, 10
}
```



# Expression Régulières - regex.exec

- La méthode `exec` à l'avantage sur la méthode `test` de récupérer les chaînes de caractères qui match et les groupes de capture

```
const regex = /\d+/g;
const str = '01/10/1985';

console.log(regex.exec(str)); // [ '01', index: 0];
console.log(regex.exec(str)); // [ '10', index: 3];
console.log(regex.exec(str)); // [ '1985', index: 6];
console.log(regex.exec(str)); // null

console.log(/(\d+)/\1\2/.exec(str));
// [ '01/10/1985', '01', '10', '1985', index: 0 ]
```



# Expression Régulières - string.match

- › La méthode *match* d'une chaîne de caractère permet de récupérer le contenu du match
- › à la différence de *test* ou *exec* elle repart au début de la chaîne de caractère à chaque recherche
- › Avec le flag *g* elle récupère tous les matchs de la regex
- › Avec des captures *groups* elle récupère le match global puis chaque capture group dans l'ordre d'apparition

```
const str = '01/10/1985';
console.log(str.match(/\d+/g)); // [ '01', '10', '1985' ]
console.log(str.match(/\d+/)); // [ '01', index: 0 ]
console.log(str.match(/(\d+)/\/(\d+)/\/(\d+)/));
// [ '01/10/1985', '01', '10', '1985', index: 0 ]
```



# Expression Régulières - string.split

- La méthode *split* d'une chaîne de caractère de la transformer en un tableau en spécifiant un séparateur
- Le séparateur peut être une chaîne de caractère ou une regex

```
var str = "Etre, ou ne pas être :\n"+  
         "telle est la question.";  
  
console.log(str.split(' '));  
// [ 'Etre', 'ou', 'ne', 'pas', 'être', ':\\ntelle', 'est', 'la', 'question.' ]  
  
console.log(str.split(/[\s,:.]/).filter(v => v));  
// [ 'Etre', 'ou', 'ne', 'pas', 'être', 'telle', 'est', 'la', 'question' ]
```



# Expression Régulières - string.search

- › La méthode `search` retourne la position du match dans la chaîne de caractère ou -1
- › Équivalent à `indexOf` mais avec une regex

```
var str = "Etre, ou ne pas être :\n"+  
         "telle est la question.";  
  
console.log(str.search(/:\n/)); // 21
```



# Expression Régulières - string.replace

- › La méthode *replace* permet de remplacer des morceaux d'une chaîne de caractères
- › Si on lui passe une regex avec des capture groups, on peut les réutiliser pour créer la nouvelle chaîne de caractères

```
var str = '01/10/1985';
var regex = /(\d+)/\/(\d+)/\/(\d+)/;

console.log(str.replace(regex, '$3-$2-$1')); // 1985-10-01
```



**formation.tech**

# ECMAScript 5.1

# ECMAScript 5.1 - Introduction



- Après ECMAScript 3, le groupe ECMAScript avance sur une nouvelle version, ECMAScript 4 qui inclut notamment les classes et les types.
- ES4 sera supporté par ActionScript (AS3) mais jamais par les navigateurs qui travaillent à une version 3.1 qui s'appellera 5 puis 5.1 après corrections pour ne pas prêter à confusion.
- Compatibilité  
CH13+, FF4+, SF5.1+, OP11.6+, IE9+ (10+ pour le mode strict, 8+ pour l'objet global JSON)  
<http://kangax.github.io/compat-table/es5/>
- Aperçu des nouvelles fonctionnalités  
<https://dev.opera.com/articles/introducing-ecmascript-5-1/>

# ECMAScript 5.1 - Mode Strict



- › Le mode strict est un mode d'exécution apparu en ECMAScript 5.1 qui vient limiter un certain nombre de mauvaises pratiques ou de problèmes de sécurité.
- › Par opposition au mode strict (strict mode), on parle parfois de sloppy mode  
[https://developer.mozilla.org/en-US/docs/Glossary/Sloppy\\_mode](https://developer.mozilla.org/en-US/docs/Glossary/Sloppy_mode)

# ECMAScript 5.1 - Mode Strict



## ▶ Activer le mode strict

- Globalement

```
'use strict';
// ... code strict...
```

- A partir d'une ligne

```
// ... code sloppy ...
'use strict';
// ... code strict...
```

- Dans une fonction

```
(function () {
  'use strict';
  // ... code strict ...
})();
```

# ECMAScript 5.1 - Mode Strict



## ► Mots clés réservés

- Sloppy Mode

```
var let = 'Hello';
console.log(let);
```

- Strict Mode

```
'use strict';

var let = 'Hello'; // SyntaxError: Unexpected strict mode reserved word
console.log(let);
```

# ECMAScript 5.1 - Mode Strict



## ▶ Oubli du mot clé var

- Sloppy Mode

```
(function() {  
    // firstName est globale  
    firstName = 'Romain';  
}());  
  
console.log(firstName); // Romain
```

- Strict Mode

```
(function() {  
    'use strict';  
    // ReferenceError: firstName is not defined  
    firstName = 'Romain';  
  
    // ReferenceError: i is not defined  
    for (i=0; i<10; i++) {}  
}());
```

# ECMAScript 5.1 - Mode Strict



## ► Désactivation de with

- Sloppy Mode

```
var int, floor = function(n) {
    return parseInt(String(n));
};

with (Math) {
    int = floor(random() * 101); // floor global ? Math.floor ?
}

console.log(int); // 42
```

- Strict Mode

```
'use strict';

var entier, floor = function(n) {
    return parseInt(String(n));
};

with (Math) { // SyntaxError: Strict mode code may not include a with statement
    entier = floor(random() * 101);
}

console.log(entier); // 42
```

# ECMAScript 5.1 - Mode Strict



## ▶ Pas d'identifiant dans eval

- Sloppy Mode

```
eval('var sum = 1 + 2');
console.log(sum); // 3
```

- Strict Mode

```
'use strict';
eval('var sum = 1 + 2');
console.log(sum); // ReferenceError: sum is not defined
```

# ECMAScript 5.1 - Mode Strict



## ▶ Supprimer des variables

- Sloppy Mode

```
var firstName = 'Romain';
var contact = {
  firstName: 'Romain'
};

delete contact.firstName;
console.log(contact.firstName); // undefined

delete firstName;
console.log(firstName); // Romain
```

- Strict Mode

```
'use strict';

var firstName = 'Romain';
var contact = {
  firstName: 'Romain'
};

delete contact.firstName;
console.log(contact.firstName); // undefined

delete firstName; // SyntaxError: Delete of an unqualified identifier in strict mode.
console.log(firstName); // Romain
```

# ECMAScript 5.1 - Mode Strict



## ▶ Utilisation de this

- Sloppy Mode

```
var Contact = function(firstName) {
  this.firstName = firstName;
};

var contact = Contact('Romain');

console.log(global.firstName); // Romain (Node.js)
console.log(window.firstName); // Romain (Browser)
```

- Strict Mode

```
'use strict';

var Contact = function(firstName) {
  this.firstName = firstName; // TypeError: Cannot set property 'firstName' of
                             // undefined
};

var contact = Contact('Romain');

console.log(global.firstName); // undefined
console.log(window.firstName); // undefined
```



# ECMAScript 5.1 - Immutable globals

- Nouvelles variables globales non modifiables

```
console.log(undefined);
console.log(NaN);
console.log(Infinity);
```

# ECMAScript 5.1 - Array



## ▶ Programmation fonctionnelle

Paradigme de programmation dans lequel les fonctions ont un rôle central et viennent remplacer les concepts de programmation impérative comme les variables, boucles, etc...

## ▶ Tableaux

Le type Array contient depuis ES5 quelques fonction qui permettent ce type de programmation (filter, map, sort, reverse, reduce, forEach...)

```
var firstNames = ['Eric', 'Romain', 'Jean', 'Eric', 'Jean'];

firstNames
  .filter(firstName => firstName.length === 4) // filtre ceux de 4 lettres
  .map(firstName => firstName.toUpperCase()) // transforme en majuscule
  .sort() // trie croissant
  .reverse() // inverse l'ordre
  .reduce((firstNames, firstName) => { // dédoublone
    if (!firstNames.includes(firstName)) {
      firstNames.push(firstName)
    }
    return firstNames;
  }, [])
  .forEach(firstName => console.log(firstName)); // affiche

// Outputs :
// JEAN
// ERIC
```



# ECMAScript 5.1 - Function.prototype.bind

- La méthode bind d'une fonction retourne une nouvelle fonction sur laquelle sera liée une nouvelle valeur this

```
var contact = {  
  firstName: 'Romain'  
};  
  
var hello = function() {  
  return 'Hello my name is ' + this.firstName;  
};  
  
console.log(hello()); // Hello my name is undefined  
var helloContact = hello.bind(contact);  
console.log(helloContact()); // Hello my name is Romain
```

# ECMAScript 5.1 - JSON



- JavaScript depuis ECMAScript 5 fourni l'objet global JSON qui contient 2 méthodes, parse (désérialiser) et stringify (sérialiser)

```
var contact = {  
    prenom: 'Romain',  
    nom: 'Bohdanowicz'  
};  
  
var json = JSON.stringify(contact);  
console.log(json); // {"prenom":"Romain","nom":"Bohdanowicz"}  
  
var object = JSON.parse(json);  
console.log(object.prenom); // Romain
```



# ECMAScript 5.1 - Trailing commas

- Il est désormais possible de placer une virgule après le dernier élément d'un tableau ou d'un objet.

```
var firstNames = [
  'Romain',
  'Jean',
  'Eric',
];

var coords = {
  x: 10,
  y: 20,
};
```



# ECMAScript 5.1 - get syntax

- › On peut masquer une méthode derrière une propriété en lecture

```
var contact = {  
    firstName: 'Romain',  
    lastName: 'Bohdanowicz',  
    get fullName() {  
        return this.firstName + ' ' + this.lastName;  
    }  
};  
  
console.log(contact.fullName); // Romain Bohdanowicz
```

# ECMAScript 5.1 - set syntax



- On peut également masquer une méthode derrière l'écriture d'une propriété

```
var contact = {
  firstName: 'John',
  lastName: 'Doe',
  set fullName(fullName) {
    var parts = fullName.split(' ');
    this.firstName = parts[0];
    this.lastName = parts[1];
  }
};

contact.fullName = 'Romain Bohdanowicz';
console.log(contact.firstName); // Romain
console.log(contact.lastName); // Bohdanowicz
```

# ECMAScript 5.1 - Object.getPrototypeOf



- ▶ `Object.getPrototypeOf` permet de retrouver le prototype d'un objet déjà instancié

```
var Person = function (firstName) {
    this.firstName = firstName;
};

Person.prototype.hello = function () {
    return 'Hello my name is ' + this.firstName;
};

var instructor = new Person('Romain');
console.log(Object.getPrototypeOf(instructor)); // Person { hello: [Function] }
console.log(Person.prototype); // Person { hello: [Function] }
```



# ECMAScript 5.1 - Object.defineProperty

- ▶ Permet une définition plus fine d'une propriété

```
var contact = { firstName: 'Romain' };

Object.defineProperty(contact, 'lastName', {
  value: 'Bohdanowicz',
  writable: false,
  enumerable: false,
  configurable: false
});

// writable: false
contact.lastName = 'Doe';
console.log(contact.lastName); // Bohdanowicz

// enumerable: false
for (var prop in contact) {
  console.log(prop); // firstName
}

// enumerable: false
console.log(JSON.stringify(contact)); // {"firstName":"Romain"}

// configurable: false
try {
  Object.defineProperty(contact, 'lastName', { value: 'Doe' });
}
catch (e) {
  console.log(e.message); // Cannot redefine property: lastName
}
```



# ECMAScript 5.1 - Object.defineProperty

- En mode strict, une propriété en lecture seule lance une exception en écriture.

```
'use strict';

var contact = {
  firstName: 'Romain'
};

Object.defineProperty(contact, 'lastName', {
  value: 'Bohdanowicz',
  writable: false,
  enumerable: false,
  configurable: false
});

// writable: false
try {
  contact.lastName = 'Doe';
}
catch (e) {
  console.log(e.message); // Cannot assign to read only property 'lastName' of
object '#<Object>'
}
```



# ECMAScript 5.1 - Object.defineProperty

- On peut masquer des méthodes derrière des propriétés en lecture/écriture

```
var contact = {
  firstName: 'Romain',
  lastName: 'Bohdanowicz'
};

Object.defineProperty(contact, 'fullName', {
  set: function(fullName) {
    var parts = fullName.split(' ');
    this.firstName = parts[0];
    this.lastName = parts[1];
  },
  get: function() {
    return this.firstName + ' ' + this.lastName;
  }
});

console.log(contact.fullName); // Romain Bohdanowicz

contact.fullName = 'John Doe';
console.log(contact.firstName); // John
console.log(contact.lastName); // Doe
```



# ECMAScript 5.1 - Object.keys

- Object.keys permet de lister les propriétés propres et énumérables

```
var Person = function (firstName) {  
    this.firstName = firstName;  
};  
  
Person.prototype.hello = function () {  
    return 'Hello my name is ' + this.firstName;  
};  
  
var instructor = new Person('Romain');  
console.log(Object.keys(instructor)); // [ 'firstName' ]
```



# ECMAScript 5.1 - Object.preventExtensions

- Il est possible d'empêcher l'extension d'un objet

```
var contact = {  
    firstName: 'Romain'  
};  
  
Object.preventExtensions(contact);  
console.log(Object.isExtensible(contact)); // false  
  
contact.name = 'Bohdanowicz';  
console.log(contact.name); // undefined
```

# ECMAScript 5.1 - Object.preventExtensions



- En mode strict, écrire dans un objet non-extensible provoque une exception

```
'use strict';

var contact = {
  firstName: 'Romain'
};

Object.preventExtensions(contact);
console.log(Object.isExtensible(contact)); // false

contact.name = 'Bohdanowicz';
console.log(contact.name); // TypeError: Can't add property name, object is not extensible
```



# ECMAScript 5.1 - Verrous

- Résumé des appels aux méthodes `Object.preventExtensions`, `Object.seal` et `Object.freeze`

Function	L'objet devient non extensible	configurable à false sur chaque propriété	writable à false sur chaque propriété
<code>Object.preventExtensions</code>	Oui	Non	Non
<code>Object.seal</code>	Oui	Oui	Non
<code>Object.freeze</code>	Oui	Oui	Oui

# ECMAScript 5.1 - Héritage en ES5



- Grâce à `Object.create`, l'héritage se fait sans dupliquer les propriétés dans le prototype.

```
var Instructor = function (firstName, speciality) {
    Person.apply(this, arguments); // héritage des propriétés de l'objet (recopie dynamique)
    this.speciality = speciality;
};

Instructor.prototype = Object.create(Person.prototype); // héritage du type et des méthodes
Instructor.prototype.constructor = Instructor;

// Redéfinition de méthode
Instructor.prototype.hello = function () {
    // Appel de la méthode parent
    return Person.prototype.hello.call(this) + ', my speciality is ' +
this.speciality;
};

var instructor = new Instructor('Romain', 'JavaScript');

console.log(instructor.hello()); // Hello my name is Romain
console.log(typeof instructor); // object
console.log(instructor instanceof Object); // true
console.log(instructor instanceof Person); // true
console.log(instructor instanceof Instructor); // true
console.log(instructor.constructor);
```



**formation.tech**

# ECMAScript 6 / ECMAScript 2015

# ECMAScript 6 - Introduction



- ECMAScript 6, aussi connu sous le nom ECMAScript 2015 ou ES6 est la plus grosse évolution du langage depuis sa création (juin 2015)  
<http://www.ecma-international.org/ecma-262/6.0/>
- Le langage est enfin adapté à des application JS complexes (modules, promesses, portées de blocks...)
- Pour découvrir les nouveautés d'ECMAScript 2015 / ES6  
<http://es6-features.org/>

# ECMAScript 6 - Compatibilité



- Compatibilité (novembre 2016) :
  - Dernière version de Chrome/Opera, Edge, Firefox, Safari : ~ 90%
  - Node.js 6 et 7 : ~ 90% d'ES6
  - Internet Explorer 11 : ~ 10% d'ES6
- Pour connaître la compatibilité des moteurs JS :  
<http://kangax.github.io/compat-table/>
- Pour développer dès aujourd'hui en ES6 et exécuter le code sur des moteurs plus anciens on peut utiliser des :
  - Compilateurs ou transpilateurs : Babel, Traceur, TypeScript... Transforment la syntaxe ES6 en ES5
  - Bibliothèques de polyfills : core-js, es6-shim, es7-shim... Recréent les méthodes manquante en JS

# ECMAScript 6 - Portées de bloc



## ▶ let

- On peut remplacer le mot-clé var, par let et obtenir ainsi une portée de bloc
- La portée de bloc ainsi créée peut devenir une closure

```
for (var globalI=0; globalI<3; globalI++) {}
console.log(typeof globalI); // number

for (let i=0; i<3; i++) {}
console.log(typeof i); // undefined

// In 1s : 0 1 2
for (let i=0; i<3; i++) {
  setTimeout(() => {
    console.log(i);
  }, 1000);
}
```

# ECMAScript 6 - Portées de bloc



- ▶ Fonction avec une portée de bloc
  - La portée de bloc s'applique également aux fonction en mode strict

```
'use strict';

if (true) {
  function test() {}
  console.log(typeof test); // function
}

console.log(typeof test); // undefined
```

# ECMAScript 6 - Constantes



## ▶ Constantes

- Il est désormais possible de créer des constantes
- Comme pour let, les variables déclarées via const ont une portée de bloc
- Bonne pratique, utiliser const ou bien let lorsque ce n'est pas possible (plus jamais var)

```
if (true) {  
    const PI = 3.14;  
}  
  
console.log(typeof PI); // undefined  
  
const hello = function() {};  
// SyntaxError: Identifier 'hello' has already been declared  
const hello = function() {};
```

# ECMAScript 6 - Template literal



- ▶ Template literal / Template string
  - Permet de créer une chaîne de caractères à partir de variables ou d'expressions
  - Permet de créer des chaînes de caractères multi-lignes
  - Déclarée avec un backquote ` (rarement utilisé dans une chaîne)

```
const prenom = 'Romain';
console.log(`Bonjour ${prenom} !`);

// ES5
// console.log('Bonjour ' + prenom + ' !');

const html =
<table class="table">
  <tr><td>${prenom.toUpperCase()}</td></tr>
</table>
`;
```

# ECMAScript 6 - new.target



## ▶ new.target

- Lors de l'appel d'une fonction, les variables this et arguments sont créées
- En ES6 il y a également super et new.target, qui est une référence vers la fonction appelante si elle est appelée avec l'opérateur new, sinon undefined

```
const Contact = function() {
  if (new.target === undefined) {
    throw new Error('Contact is a constructor');
  }
};

const c1 = new Contact(); // OK
const c2 = Contact(); // Error: Contact is a constructor
```

# ECMAScript 6 - Fonctions fléchées



## ▶ Arrow Functions

- Plus courtes à écrire : (params) => retour.
- Si un seul paramètre, les parenthèses des paramètres sont optionnelles.
- Si le retour est un objet, les parenthèses du retour sont obligatoires.

```
const sum = (a, b) => a + b;
const hello = name => `Hello ${name}`;
const getCoords = (x, y) => ({x: x, y: y});

// ES5
// var sum = function (a, b) {
//   return a + b;
// };
// var hello = function (name) {
//   return 'Hello ' + name;
// };
// var getCoords = function (x, y) {
//   return {
//     x: x,
//     y: y,
//   };
// };
```

# ECMAScript 6 - Fonctions fléchées



## ▶ Avec bloc d'instructions

- Si les fonctions nécessitent plusieurs lignes, on peut utiliser un bloc {}
- Le mot clé return devient alors obligatoire

```
const isWon = (nbGiven, nbToGuess) => {
  if (nbGiven < nbToGuess) {
    return 'Too low';
  }

  if (nbGiven > nbToGuess) {
    return 'Too high';
  }

  return 'Won !';
};
```

# ECMAScript 6 - Fonctions fléchées



## ▶ Bonnes pratiques

- Attention à ne pas utiliser les fonctions fléchées pour déclarer des méthodes !
- Utiliser les fonctions fléchées pour les callback ou les fonctions hors objets
- Utiliser les method properties pour les méthodes
- Utiliser class pour les fonctions constructeurs

```
const globalThis = this;

const contact = {
  firstName: 'Romain',
  method1: () => { // Mauvaise pratique
    console.log(this === globalThis); // true
  },
  method2() { // Bonne pratique
    console.log(this === contact); // true
  }
};

contact.method1();
contact.method2();
```

# ECMAScript 6 - Default Params



## ▶ Paramètres par défaut

- Les paramètres d'entrées peuvent maintenant recevoir une valeur par défaut

```
const sum = function(a, b, c = 0) {
  return a + b + c;
};

console.log(sum(1, 2, 3)); // 6
console.log(sum(1, 2)); // 3

// ES5
// var sum = function(a, b, c) {
//   if (c === undefined) {
//     c = 0;
//   }
//   return a + b + c;
// };
```

# ECMAScript 6 - Default Params



- ▶ Paramètres par défaut
  - Les valeurs par défaut sont calculées au moment de l'appel et peuvent être des appels de fonctions

```
const frDate = function(date = new Date()) {  
    return date.toLocaleDateString();  
};  
  
console.log(frDate(new Date('1985-10-01'))); // 01/10/1985  
console.log(frDate()); // 26/11/2017
```

# ECMAScript 6 - Rest Parameters



## ▶ Paramètres restants

- Pour récupérer les valeurs non déclarées d'une fonction on peut utiliser le REST Params
- Remplace la variable arguments (qui n'existe pas dans une fonction fléchée)
- La variable créée est un tableau (contrairement à arguments)
- Bonne pratique : ne plus utiliser arguments

```
const sum = (a, b, ...others) => {
  let result = a + b;

  others.forEach(nb => result += nb);

  return result;
};
console.log(sum(1, 2, 3, 4)); // 10

const sumShort = (...n) => n.reduce((a, b) => a + b);
console.log(sumShort(1, 2, 3, 4)); // 10
```

# ECMAScript 6 - Spread Operator



## ► Spread Operator

- Le Spread Operator permet de transformer un tableau en une liste de valeurs.

```
const sum = (a, b, c, d) => a + b + c + d;

const nbs = [2, 3, 4, 5];
console.log(sum(...nbs)); // 14
// ES5 :
// console.log(sum(nbs[0], nbs[1], nbs[2], nbs[3]));

const otherNbs = [1, ...nbs, 6];
console.log(otherNbs.join(', ')); // 1, 2, 3, 4, 5, 6
// ES5 :
// const otherNbs = [1, nbs[0], nbs[1], nbs[2], nbs[3], 6];

// Clone an array
const cloned = [...nbs];
```

# ECMAScript 6 - Shorthand property



- ▶ Shorthand property

- Lorsque l'on affecte une variable à une propriété (maVar: maVar), il suffit de déclarer la propriété

```
const x = 10;
const y = 20;

const coords = {
  x,
  y,
};

// ES5
// const coords = {
//   x: x,
//   y: y,
// };
```

# ECMAScript 6 - Method properties



- ▶ Method properties
  - Syntaxe simplifiée pour déclarer des méthodes

```
const maths = {  
    sum(a, b) {  
        return a + b;  
    }  
};  
  
console.log(maths.sum(1, 2)); // 3  
  
// ES5  
// const maths = {  
//     sum: function(a, b) {  
//         return a + b;  
//     }  
// };
```

# ECMAScript 6 - Computed Property Names



- ▶ Computed Property Names

Permet d'utiliser une expression en nom de propriété

```
let i = 0;

const users = {
  [`user${++i}`]: { firstName: 'Romain' },
  [`user${++i}`]: { firstName: 'Steven' },
};

console.log(users.user1); // { firstName: 'Romain' }

/* ES5
var i = 0;
var users = {};
users['user ' + (++i)] = { firstName: 'Romain' };
users['user ' + (++i)] = { firstName: 'Steven' };

console.log(users.user1); // { firstName: 'Romain' }
*/
```

# ECMAScript 6 - Array Destructuring



- ▶ Déstructurer un tableau
  - Permet de déclarer des variables recevant directement une valeur d'un tableau

```
const [one, two, three] = [1, 2, 3];
console.log(one); // 1
console.log(two); // 2
console.log(three); // 3

// ES5
// var tmp = [1, 2, 3];
// var one = tmp[0];
// var two = tmp[1];
// var three = tmp[2];
```

# ECMAScript 6 - Array Destructuring



- ▶ Déstructurer un tableau
  - Il est possible de ne pas déclarer une variable pour chaque valeur
  - Il est possible d'utiliser une valeur par défaut
  - Il est possible d'utiliser le REST Params

```
const [one, , three = 3] = [1, 2];
console.log(one); // 1
console.log(three); // 3

const [romain, ...others] = ['Romain', 'Jean', 'Eric'];
console.log(romain); // Romain
console.log(others.join(', ')); // Jean, Eric
```



# ECMAScript 6 - Object Destructuring

- ▶ Déstructurer un object
  - Comme pour les tableaux il est possible de déclarer une variable recevant directement une propriété

```
const {x: varX, y: varY} = {x: 10, y: 20};  
console.log(varX); // 10  
console.log(varY); // 20
```

# ECMAScript 6 - Object Destructuring



- ▶ Déstructurer un object
  - Il est possible de nommer sa variable comme la propriété et d'utiliser shorthand property
  - Il est possible d'utiliser une valeur par défaut

```
const {x: x, y, z = 30} = {x: 10, y: 20};  
console.log(x); // 10  
console.log(y); // 20  
console.log(z); // 30
```

# ECMAScript 6 - Mot clé class



- ▶ Simplifie la déclaration de fonction constructeur
- ▶ Les classes n'existent pas pour autant en JavaScript, ce n'est qu'une syntaxe simplifiée (sucre syntaxique)
- ▶ Le contenu d'une classe est en mode strict

```
class Person {  
    constructor(firstName) {  
        this.firstName = firstName;  
    }  
    hello() {  
        return `Hello my name is ${this.firstName}`;  
    }  
}  
  
const instructor = new Person('Romain');  
console.log(instructor.hello()); // Hello my name is Romain  
  
// ES5  
// var Person = function(firstName) {  
//     this.firstName = firstName;  
// };  
// Person.prototype.hello = function() {  
//     return 'Hello my name is ' + this.firstName;  
// };
```

# ECMAScript 6 - Mot clé class



- ▶ Héritage avec le mot clé class
  - Utilisation du mot clé extends pour l'héritage
  - Utilisation de super pour appeler la fonction constructeur parent et les accès aux méthodes parents si redéclarée dans la classe

```
class Instructor extends Person {  
    constructor(firstName, speciality) {  
        super(firstName);  
        this.speciality = speciality;  
    }  
    hello() {  
        return `${super.hello()}, my speciality is ${this.speciality}`;  
    }  
}  
  
const romain = new Instructor('Romain', 'JavaScript');  
console.log(romain.hello()); // Hello my name is Romain, my speciality is  
JavaScript
```



# ECMAScript 6 - Boucle for .. of

## ▶ Boucle for .. of

- Permet de boucler sur des objets itérables (Array, Map, Set, String, TypedArray, arguments)

```
const firstNames = ['Romain', 'Eric'];

for (const firstName of firstNames) {
  console.log(firstName);
}
```

# ECMAScript 6 - Object.assign



## ▶ Object.assign

- Permet d'affecter toutes les clés d'un objet à un autre objet
- Utile pour cloner une objet
- Attention le clone ne concerne pas les sous-objets ou tableaux

```
const obj = {  
  firstName: 'Romain',  
  address: {  
    city: 'Paris'  
  }  
};  
  
const copy = Object.assign({}, obj);  
console.log(copy === obj); // false  
console.log(copy.address === obj.address); // true
```

# ECMAScript 6 - Object.assign



## ▶ Object.assign

- Pour faire un clone de tous les sous objet on pourrait écrire une fonction récursive
- Ou alors utiliser la fonction cloneDeep de Lodash

```
const deepClone = function(obj) {
  let clone = Object.assign({}, obj);

  for (let p in obj) {
    if (obj.hasOwnProperty(p) && typeof obj[p] === 'object') {
      clone[p] = deepClone(obj[p]);
    }
  }

  return clone;
};

const deepCopy = deepClone(obj);
console.log(deepCopy.address === obj.address); // false
```

# ECMAScript 6 - Générateurs



- ▶ Générateurs
  - Fonction déclarée avec \*
  - Peut être retourner un résultat et se mettre en pause (yield)

```
function *nbs() {  
  let i = 0;  
  while (i < 3) {  
    yield ++i;  
  }  
}  
  
for (let i of nbs()) {  
  console.log(i); // 1 2 3  
}
```

# ECMAScript 6 - Symbol



- Symbol est un nouveau type primitif qui n'a pas de syntaxe littéral, seul l'appel à la fonction Symbol est possible
- 2 appels successifs à Symbol donneront 2 valeurs uniques

```
var locale = {  
    fr_FR: Symbol(),  
    en_US: Symbol()  
};  
  
var translations = {  
    [locale.fr_FR]: {  
        'hello': 'bonjour',  
        'cat': 'chat'  
    },  
    [locale.en_US]: {  
        'hello': 'hello',  
        'cat': 'cat'  
    }  
};  
  
var translate = function (key, locale = locales.en_US) {  
    return translations[locale][key];  
};  
  
console.log(translate('hello', locale.fr_FR)); // bonjour
```

# ECMAScript 6 - Symbol



- ▶ Symbol permet également de redéfinir des comportements du langage, comme la boucle for..of avec Symbol.iterator

```
class Collection {
  constructor() {
    this.list = [];
  }
  add(elt) {
    this.list.push(elt);
    return this;
  }
  *[Symbol.iterator]() {
    for (let elt of this.list) {
      yield elt;
    }
  }
}

let firstNames = new Collection();
firstNames.add('Romain').add('Eric');

for (let firstName of firstNames) {
  console.log(firstName); // Romain Eric
}
```

# ECMAScript 6 - Exercice



- Reprendre le jeu du plus ou moins
- Le transformer en utilisant les mots clés class, let et les fonctions fléchées



**formation.tech**

# JavaScript Asynchrone



# JavaScript Asynchrone - Introduction

## ▶ Boucle d'événement

Comme vu précédemment, le code JavaScript s'exécute au sein d'une boucle appelée « boucle d'événement ». Ceci permet de différer l'exécution d'une partie d'un code au moment où une interaction se produit (ex : clic, fin de chargement, réception de données, requêtes HTTP, lecture de fichier).

## ▶ Avantages

- Gestion de la concurrence simplifiée
- Performance

## ▶ Inconvénients

- Perte de contexte (mot clé this)
- Callback Hell



# JavaScript Asynchrone - Perte de contexte

- ▶ Où est this ?

Dans l'exemple ci-dessous on mélange code objet et programmation asynchrone. Problème, au moment où le callback est appelé (dans un prochain passage de la boucle d'événement), le moteur JavaScript perd la référence sur l'objet this qui était attaché à la méthode helloAsync.

```
var contact = {
  firstName: 'Romain',
  helloAsync: function() {
    setTimeout(function() {
      console.log('Hello my name is ' + this.firstName);
    }, 1000)
  }
};

contact.helloAsync(); // Hello my name is undefined
```



# JavaScript Asynchrone - Perte de contexte

- ▶ Dans l'implémentation setTimeout de Node.js :  
<https://github.com/nodejs/node/blob/v6.x/lib/timers.js#L382>

```
function ontimeout(timer) {
  var args = timer._timerArgs;
  var callback = timer._onTimeout;
  if (!args)
    timer._onTimeout();
  else {
    switch (args.length) {
      case 1:
        timer._onTimeout(args[0]);
        break;
      case 2:
        timer._onTimeout(args[0], args[1]);
        break;
      case 3:
        timer._onTimeout(args[0], args[1], args[2]);
        break;
      default:
        Function.prototype.apply.call(callback, timer, args);
    }
  }
  if (timer._repeat)
    rearm(timer);
}
```



# JavaScript Asynchrone - Perte de contexte

- ▶ Solution 1 : Sauvegarder this dans la portée de closure

La valeur de this peut être sauvegardée dans la portée de closure, la variable s'appelle généralement that (ou \_this, self, me...)

```
var contact = {
  firstName: 'Romain',
  helloAsync: function() {
    var that = this;
    setTimeout(function() {
      console.log('Hello my name is ' + that.firstName);
    }, 1000)
  }
};

contact.helloAsync(); // Hello my name is Romain
```



# JavaScript Asynchrone - Perte de contexte

- ▶ Solution 2 : Function.bind (ES5)

La méthode bind du type function retourne une fonction dont la valeur de this ne peut être modifiée.

```
var contact = {
  firstName: 'Romain',
  helloAsync: function() {
    setTimeout(function() {
      console.log('Hello my name is ' + this.firstName);
    }.bind(this), 1000)
  }
};
```

```
contact.helloAsync(); // Hello my name is Romain
```

```
var contact = {
  firstName: 'Romain',
  hello: function() {
    console.log('Hello my name is ' + this.firstName);
  },
  helloAsync: function() {
    setTimeout(this.hello.bind(this), 1000);
  }
};
```

```
contact.helloAsync(); // Hello my name is Romain
```

# JavaScript Asynchrone - Perte de contexte



- ▶ Solution 3 : Arrow Function (ES6)

Les fonctions fléchées ne lient pas de valeur pour this, ce qui permet au callback de retrouvé la valeur de la fonction parent.

```
var contact = {  
    firstName: 'Romain',  
    helloAsync() {  
        setTimeout(() => {  
            console.log('Hello my name is ' + this.firstName);  
        }, 1000)  
    }  
};  
  
contact.helloAsync(); // Hello my name is Romain
```



# JavaScript Asynchrone - Callback Hell

- › Callback Hell / Pyramid of doom

Devoir attendre que le callback soit appelé pour démarrer l'opération suivante oblige à imbriquer le code qui lui

```
const fs = require('fs');

fs.readFile('./src/index.html', (err, buffer) => {
  if (err) {
    return console.log(err);
  }
  fs.writeFile('./dist/index.html', buffer, (err) => {
    if (err) {
      return console.log(err);
    }
    console.log('File copied');
  });
});
```

# JavaScript Asynchrone - Callback Hell



## › Callback Hell

A force le code JavaScript a tendance à s'imbriquer, ici une simple copie de fichier nécessite de lire le fichier de manière asynchrone puis de l'écrire.

```
const fs = require('fs');
const path = require('path');

const file = 'index.html';
const distDirPath = path.join(__dirname, 'dist');
const srcDirPath = path.join(__dirname, 'src');
const srcFilePath = path.join(srcDirPath, file);
const distFilePath = path.join(distDirPath, file);

fs.readFile(srcFilePath, (err, data) => {
  if (err) {
    return console.log(err);
  }
  fs.writeFile(distFilePath, data, (err) => {
    if (err) {
      return console.log(err);
    }
    console.log(`File ${file} copied.`);
  });
});
```

# JavaScript Asynchrone - Async



## ▶ Async

La bibliothèque Async contient un certain nombre de méthodes pour simplifier les problématiques d'asynchronisme, ici waterfall appelle le premier callback, passe le résultat au second puis appelle le dernier callback, ou directement le dernier en cas d'erreur.

```
const fs = require('fs');
const path = require('path');
const async = require('async');

const file = 'index.html';
const distDirPath = path.join(__dirname, 'dist');
const srcDirPath = path.join(__dirname, 'src');
const srcFilePath = path.join(srcDirPath, file);
const distFilePath = path.join(distDirPath, file);

async.waterfall([
  (callback) => fs.readFile(srcFilePath, callback),
  (data, callback) => fs.writeFile(distFilePath, data, callback),
], (err) => {
  if (err) {
    return console.log(err);
  }
  console.log(`File ${file} copied.`);
});
```

# JavaScript Asynchrone - Promesses



## ► Exemple avancé

Les promesses sont un concept pas si nouveau en JavaScript, on les retrouve dans jQuery depuis la version 1.5 (deferred object).

Elle permet de gagner en lisibilité en remettant à plat un code asynchrone, tout en offrant la possibilité à du code asynchrone d'utiliser les exceptions.

On peut les utiliser grâce à des bibliothèques comme bluebird ou q, ou bien nativement depuis ES6.

```
const fs = require('fs');
const path = require('path');

const file = 'index.html';
const distDirPath = path.join(__dirname, 'dist');
const srcDirPath = path.join(__dirname, 'src');
const srcFilePath = path.join(srcDirPath, file);
const distFilePath = path.join(distDirPath, file);

fs.promises.readFile(srcFilePath)
  .then((content) => fs.promises.writeFile(distFilePath, content))
  .then(() => console.log(`File ${file} copied.`))
  .catch(console.log);
```



# JavaScript Asynchrone - Promesses

- Exemple avancé  
5 callbacks imbriqués et une gestion d'erreur intermédiaire puis finale avec les promesses

```
const fsp = require('fs-promise');
const path = require('path');

const file = 'index.html';
const distDirPath = path.join(__dirname, 'dist');
const srcDirPath = path.join(__dirname, 'src');
const srcFilePath = path.join(srcDirPath, file);
const distFilePath = path.join(distDirPath, file);

fsp.stat(distDirPath)
  .catch(err => fsp.mkdir(distDirPath))
  .then(() => fsp.readFile(srcFilePath))
  .then(content => fsp.writeFile(distFilePath, content))
  .then(() => console.log(`File ${file} copied.`))
  .catch(console.log);
```

# JavaScript Asynchrone - Observables



- ▶ Promise
  - Utilisable uniquement pour des événements ponctuels (ex: setTimeout)
  - Pas annulable (sauf en utilisant des bibliothèques comme bluebird)
- ▶ Observable
  - Utilisable pour des événements ponctuels ou récurrents (ex: setInterval)
  - Annulable
  - Contient des opérateurs pour l'enrichir (map, debounce, flatMap...)
  - Peut se créer et se déclencher en 2 temps

# JavaScript Asynchrone - Observables



- Exemple Promise vs Observable

```
const Observable = require('rxjs').Observable;

const timeout = new Promise((resolve) => {
  setTimeout(() => {
    resolve();
  }, 1000);
});

timeout.then(() => {
  console.log('timeout 1s');
});

const interval$ = new Observable((observer) => {
  setInterval(() => {
    observer.next();
  }, 1000);
});

interval$.subscribe(() => {
  console.log('observable 1s');
});
```



# JavaScript Asynchrone - Exercice

- Exercice de build  
Ennoncé sur <https://github.com/bioub/Exercice-Build>
- A faire avec Promise et éventuellement async / await



**formation.tech**

# ECMAScript 7 / ECMAScript 2016

# ECMAScript 7 - Introduction



- ECMAScript 7 sort en juin 2016 et ne contient que 2 nouveautés
  - 1 nouvelle syntaxe : Opérateur d'exponentiation
  - 1 nouvel API : Array.prototype.includes

# ECMAScript 7 - Opérateur d'exponentiation



## ► Opérateur d'exponentiation

Renvoie le résultat de l'élévation d'un nombre (premier opérande) à une puissance donnée (deuxième opérande).

Par exemple : var1 \*\* var2 sera équivalent à  $\text{var1}^{\text{var2}}$  en notation mathématique.

```
console.log(2 ** 3); // 8  
  
// Équivalent en ES6 à  
console.log(Math.pow(2, 3)); // 8
```

## ► Plugin babel : transform-exponentiation-operator

<https://babeljs.io/docs/plugins/transform-exponentiation-operator/>

# ECMAScript 7 - Array.prototype.includes



## ▶ Array.prototype.includes

L'API Array introduit une nouvelle méthode pour vérifier si un élément est présent dans un tableau.

Attention pour les objets (sauf string), includes vérifie les références et non le contenu de l'objet.

```
const nbs = [2, 3, 4];

console.log(nbs.includes(2)); // true
console.log(nbs.includes(5)); // false

const contacts = [{prenom: 'Romain'}];
console.log(contacts.includes({prenom: 'Romain'})); // false
```



**formation.tech**

# ECMAScript 8 / ECMAScript 2017

# ECMAScript 8 - Introduction



- ▶ ECMAScript 8 sort en juin 2017 et contient 4 nouveautés
  - 2 nouvelles syntaxes :
    - Virgules finales sur les fonctions
    - Fonctions async / await
  - 3 nouveaux APIs
    - Object.values / Object.entries
    - String Padding
    - Atomics

# ECMAScript 8 - Virgules finales sur les fonctions



## ▶ Virgules finales sur les fonctions

Comme pour object literal et array literal, il est désormais possible que le dernier argument d'une fonction soit suivi d'une virgule (au moment de l'appel ou de la déclaration)

```
console.log(  
  'A very very very very very loooooooooooooonnnngggg string',  
  'Lorem ipsum dolor sit amet, consectetur adipiscing elit.',  
  'New line',  
);
```

```
class ContactsComponent {  
  constructor(  
    contactService,  
    logService,  
    httpClient,  
  ) {  
    this.contactService = contactService;  
    this.logService = logService;  
    this.httpClient = httpClient;  
  }  
}
```



# ECMAScript 8 - Virgules finales sur les fonctions

- Depuis ES5 c'était déjà possible sur object et array literal
  - Array literal

```
const firstNames = [  
  'Romain',  
  'Jean',  
  'Eric',  
];
```

- Object literal

```
const coords = {  
  x: 10,  
  y: 20,  
};
```



# ECMAScript 8 - Virgules finales sur les fonctions

- Exemple de diff sans et avec virgule finale

```
$ git diff trailing-commas.js
diff --git a/trailing-commas.js b/trailing-commas.js
index da942a9..9064804 100644
--- a/trailing-commas.js
+++ b/trailing-commas.js
@@ -1,4 +1,5 @@
console.log(
  'A very very very very very loooooooooonnnngggg string',
- 'Lorem ipsum dolor sit amet, consectetur adipiscing elit.'
+ 'Lorem ipsum dolor sit amet, consectetur adipiscing elit.',
+ 'New line',
);
```

```
$ git diff trailing-commas.js
diff --git a/trailing-commas.js b/trailing-commas.js
index 32f26a2..6ccd800 100644
--- a/trailing-commas.js
+++ b/trailing-commas.js
@@ -1,4 +1,5 @@
console.log(
  'A very very very very very loooooooooonnnngggg string',
  'Lorem ipsum dolor sit amet, consectetur adipiscing elit.',
+ 'New line',
);
```

# ECMAScript 8 - Fonctions async / await



- ▶ **async / await**

Permet d'écrire du code basé sur des promesses comme on aurait écrit du code synchrone.

- ▶ Soit la fonction timeout suivante (qui retourne une Promesse)

```
const timeout = (delay) => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (delay % 1000 !== 0) {
        return reject(new Error('delay must be a multiple of 1000'));
      }
      resolve();
    }, delay);
  });
};
```



# ECMAScript 8 - Fonctions async / await

## ► Sans async / await

```
timeout(1000)
  .then(() => {
    console.log('1s');
    return timeout(1000);
  })
  .then(() => {
    console.log('2s');
    return timeout(500);
  })
  .then(() => console.log('2.5s'))
  .catch(err => console.log(`Error : ${err.message}`));
```

## ► Avec async / await

```
(async () => {
  try {
    await timeout(1000);
    console.log('1s');
    await timeout(1000);
    console.log('2s');
    await timeout(500);
  }
  catch (err) {
    console.log(`Error : ${err.message}`);
  }
})();
```



# ECMAScript 8 - Object.values / Object.entries

## ▶ Object.values

Retourne les valeurs d'un objet sous la forme d'un tableau

```
const contact = {
  firstName: 'Romain',
  lastName: 'Bohdanowicz',
};

for (const value of Object.values(contact)) {
  console.log('value', value); // Romain, Bohdanowicz
}
```

## ▶ Object.entries

Retourne les clés/valeurs d'un objet sous la forme d'un tableau de tableau [key, value]

```
const contact = {
  firstName: 'Romain',
  lastName: 'Bohdanowicz',
};

for (const [key, value] of Object.entries(contact)) {
  console.log('key', key); // firstName, lastName
  console.log('value', value); // Romain, Bohdanowicz
}
```



# ECMAScript 8 - String Padding

## ▶ String Padding

Permet d'obtenir une chaîne de caractère sur un nombre de caractères fixes en la complétant au début (padStart) ou à la fin (padEnd) par une chaîne de caractère de son choix.

```
console.log('7'.padStart(3, '0')); // 007
console.log('Titre'.padEnd(20, '.')); // Titre.....
console.log('Woohoo'.padEnd(20, '0o'))); // Woohoo0o0o0o0o0o0o0o0o
```

# ECMAScript 8 - Atomics



- ▶ Méthodes statiques de l'objet Atomics

Lorsque la mémoire est partagée, plusieurs threads peuvent lire et écrire sur les mêmes données en mémoire. Les opérations atomiques permettent de s'assurer que des valeurs prévisibles sont écrites et lues, que les opérations sont finies avant que la prochaine débute et que les opérations ne sont pas interrompues.

```
// create a SharedArrayBuffer with a size in bytes
var buffer = new SharedArrayBuffer(16);
var uint8 = new Uint8Array(buffer);
uint8[0] = 7;

// 7 + 2 = 9
console.log(Atomics.add(uint8, 0, 2));
// expected output: 7

console.log(Atomics.load(uint8, 0));
// expected output: 9
```

# ECMAScript 8 - Object.getOwnPropertyDescriptors



- La méthode `Object.getOwnPropertyDescriptors()` renvoie l'ensemble des descripteurs des propriétés propres d'un objet donné.

```
const contact = {
  firstName: 'Romain',
};

Object.defineProperty(contact, '_visible', {
  value: true,
  enumerable: false,
});

const descriptors = Object.getOwnPropertyDescriptors(contact);
console.log(descriptors.firstName.writable); // true
console.log(descriptors._visible.enumerable); // false
```



**formation.tech**

# ECMAScript 9 / ECMAScript 2018

# ECMAScript 9 - Rest/Spread Properties



- Rest/Spread Properties  
Syntaxes inspirée de Rest/Spread sur les tableaux
- Rest  
Permet de récupérer les propriétés restantes
- Spread  
Permet d'affecter l'ensemble des propriétés à un autre objet

```
const coords = {  
    x: 10,  
    y: 20,  
    z: 30,  
};  
  
// Rest  
const {z, ...coords2d} = coords;  
console.log(JSON.stringify(coords2d)); // {"x":10,"y":20}  
  
// Spread  
const coords3d = {...coords2d, z};  
console.log(JSON.stringify(coords3d)); // {"x":10,"y":20,"z":30}  
  
const clone = {...coords};
```

# ECMAScript 9 - Template Literal Revision



- Modifie la spec Template Literal pour qu'elle autorise des séquences commençant par \u, \x, \0 autre que \u00FF or \u{42}, \xFF ou \0100

```
let bad = `bad escape sequence: \unicode`; // throws early error
```

# ECMAScript 9 - Regexp DotAll Flag



- Ajoute un flag aux expressions régulières pour que le meta-caractère '.' inclue les retours à la ligne
- Avant ES9 il aura fallu créer un meta-caractère `[\s\S]` (`\s` tous les caractères autres que des caractères non-imprimable, `\S` tous les caractères non imprimables)

```
const htmlStr = `<body>
  <div class="clock"></div>
  <script src="clock.js"></script>
  <script src="index.js"></script>
</body>
`;

const distTag = '<script src="bundle.js"></script>';
const regex = /<script.*</script>/s;
// ES8
// const regex = /<script[\s\S]*</script>/;

console.log(htmlStr.replace(regex, distTag));
/*
<body>
  <div class="clock"></div>
  <script src="bundle.js"></script>
</body>
*/
```

# ECMAScript 9 - Regexp Capture Group



- › Il est désormais possible de nommer les capture groups (entre parenthèse dans une regexp)
- › Les parenthèses commencent alors par ?<nom>
- › Le retour n'est plus un tableau mais un objet

```
const regex = /(?<year>\d{4})-(?<month>\d{2})-(?<day>\d{2})/;
const {groups: {year, month, day}} = regex.exec('1985-10-01');

console.log('year', year); // 1985
console.log('month', month); // 10
console.log('day', day); // 01

// ES8
// const regex = /(\d{4})-(\d{2})-(\d{2})/;
// const [, year, month, day] = regex.exec('1985-10-01');
//
// console.log('year', year); // 1985
// console.log('month', month); // 10
// console.log('day', day); // 01
```

# ECMAScript 9 - Regexp Lookbehind



- On pouvait avec des expressions régulières, matcher qui en précède une autre, sans capturer la seconde
- On peut depuis ES9 capturer la suite, sans capturer ce qui précède

```
// Lookbehind ES9
const str = 'It is two past ten';
console.log(/(?<=\spast\s)\w+/.exec(str)[0]); // ten
console.log(/(?<!\spast\s)\w+/.exec(str)[0]); // It

// Lookahead ES8
console.log(/\w+(?= \spast\s)/.exec(str)[0]); // two
console.log(/\w+(?! \spast\s)/.exec(str)[0]); // It
```

# ECMAScript 9 - Regexp Unicode property escapes



- ES6 introduit les expressions régulières unicode
- ES9 permet de spécifier des types de caractères unicode  
<http://unicode.org/reports/tr18/#Categories>

```
const frenchStr = 'Une phrase en français';
console.log(frenchStr.match(/[a-zA-Z]+/gu));
// [ 'Une', 'phrase', 'en', 'fran', 'ais' ]

console.log(frenchStr.match(/\p{Alphabetic}+/gu));
// [ 'Une', 'phrase', 'en', 'français' ]

const emojiStr = 'LOL 😂😊🤣 Awesome !!! 😜';
console.log(emojiStr.match(/\p{Emoji}/gu));
// [ '😂', '😊', '🤣', '😜' ]
```



# ECMAScript 9 - Promise.prototype.finally

- ▶ Introduction d'une méthode finally qui sera toujours exécutée, erreur ou non

```
const fs = require('fs');

let filehandle;

fs.promises.open('./example.txt', 'r')
  .then((fd) => {
    filehandle = fd;
    return filehandle.readFile();
})
  .then((data) => {
    console.log(data.toString()); // Contenu du fichier
})
  .finally(() => {
    filehandle.close();
});
```

# ECMAScript 9 - Promise.prototype.finally



- ▶ Avec les fonctions asynchrones

```
const fs = require('fs');

(async () => {
  let filehandle;
  try {
    filehandle = await fs.promises.open('./example.txt', 'r');
    const data = await filehandle.readFile();
    console.log(data.toString()); // Contenu du fichier
  }
  finally {
    filehandle.close();
  }
})();
```

# ECMAScript 9 - Asynchronous iteration



- ▶ `for - await - of` permet de boucler sur des itérateurs ou des générateurs asynchrones

```
const fs = require('fs');

async function* readLines(path, bufferSize = 4096, encoding = 'utf-8') {
  let filehandle = await fs.promises.open(path, 'r');

  try {
    let eof = false;
    let strBuffer = '';
    do {
      const { bytesRead, buffer } = await filehandle.read(Buffer.alloc(bufferSize), 0, bufferSize);
      strBuffer += buffer.toString(encoding, 0, bytesRead);
      const lines = strBuffer.split(/\n|\r\n|\r/);

      for (const line of lines.slice(0, lines.length - 1)) {
        yield line;
      }

      strBuffer = lines[lines.length - 1];
      eof = !bytesRead;
    }
    while (!eof);
  } finally {
    await filehandle.close();
  }
}

(async () => {
  for await (const line of readLines('./3-lines.txt')) {
    console.log(line);
  }
})();
```



**formation.tech**

# ECMAScript 10 / ECMAScript 2019



# ECMAScript 10 - Optional Catch Binding

- ▶ Les parenthèses d'un bloc catch deviennent optionnel lorsque l'erreur n'est pas utilisée

```
const fs = require('fs');

try {
  fs.statSync('./does-not-exists');
} catch {
  console.log('An error occurred');
}
```

# ECMAScript 10 - JSON Superset



- Ajoute le support des caractères Unicode non échappés U+2028 LINE SEPARATOR and U+2029 PARAGRAPH SEPARATOR
- Ces derniers étant présents dans la norme JSON mais indisponibles dans les chaînes de caractères JS



# ECMAScript 10 - Symbol.prototype.description

- La classe Symbol contient une nouvelle propriété description qui permet de récupérer la valeur spécifiée à la création

```
const symbol = Symbol('My Symbol!');

console.log(symbol.toString()); // Symbol(My Symbol!)
console.log(symbol.description); // My Symbol!
```



# ECMAScript 10 - Object.fromEntries

- Object.fromEntries est la méthode opposée à Object.entries

```
const coords = {x: 10, y: 20};  
const entries = Object.entries(coords);  
console.log(entries); // [ [ 'x', 10 ], [ 'y', 20 ] ]  
  
console.log(Object.fromEntries(entries)); // { x: 10, y: 20 }
```



# ECMAScript 10 - JSON.stringify

- JSON.stringify n'essaie plus de créer des représentation pour des séquences Unicode qui sont impossibles

```
console.log(JSON.stringify('\uD834\uDF06'));
// ≡

console.log(JSON.stringify('\uDF06\uD834'));
// \udf06\ud834 (après ES10)
// (avant ES10)
```



# ECMAScript 10 - trimStart / trimEnd

- ES5 a normé String.prototype.trim
- Les principaux moteurs JS ont également implémenté les fonctions trimLeft and trimRight - sans qu'une norme existe.
- Par consistence avec padStart et padEnd, ES10 norme 2 fonction trimStart / trimEnd ainsi que les fonctions trimLeft et trimRight pour ne pas casser les sites existants

```
console.log(' abc '.trimStart()); // 'abc '
console.log(' abc '.trimLeft()); // 'abc '
console.log(' abc '.trimEnd()); // 'abc'
console.log(' abc '.trimRight()); // 'abc'
```

# ECMAScript 10 - flat / flatMap



- Array.prototype.flat permet d'aplatir un tableau
- On peut lui spécifier une profondeur (1 par défaut)
- La méthode peut être combinée avec map en appelant flatMap directement

```
const nbs = [1, [2, [3]]];

console.log(nbs.flat()); // [ 1, 2, [ 3 ] ]
console.log(nbs.flat(2)); // [ 1, 2, 3 ]
console.log(nbs.flat(Infinity)); // [ 1, 2, 3 ]

const lts = ['a', 'b', 'c'];

console.log(lts.map((lt) => [lt, lt]).flat());
// [ 'a', 'a', 'b', 'b', 'c', 'c' ]

console.log(lts.flatMap((lt) => [lt, lt]));
// [ 'a', 'a', 'b', 'b', 'c', 'c' ]
```



**formation.tech**

# ECMAScript 11 / ECMAScript 2020

# ECMAScript 11 - String.prototype.matchAll



- › La méthode *match* d'un objet string permet de récupérer un élément de la chaîne de caractère qui correspond à une expression régulière
- › Lorsque qu'on ajoute le flag g à l'expression régulière, on perd la capture des groupes (les parenthèses de l'expression)
- › La nouvelle méthode *matchAll* permet de combiner les 2

```
const str = 'du 01/01/2020 au 30/06/2020';

console.log(str.match(/(\d{2})\/(\d{2})\/(\d{4})/));
// [ '01/01/2020', '01', '01', '2020']

console.log(str.match(/(\d{2})\/(\d{2})\/(\d{4})/g));
// [ '01/01/2020', '30/06/2020' ]

console.log(Array.from(str.matchAll(/(\d{2})\/(\d{2})\/(\d{4})/g)));
/*
[
  [ '01/01/2020', '01', '01', '2020'],
  [ '30/06/2020', '30', '06', '2020']
]
```

# ECMAScript 11 - import()



- › La fonction import permet d'inclure un module ECMAScript de manière dynamique et asynchrone
- › La fonction retourne une promesse
- › Les bundlers comme webpack sont déjà compatibles et mettront le code à importer dans des fichiers supplémentaires, permettant de différer leur chargement

```
// my-math.js
export const sum = (a, b) => a + b;
export const sub = (a, b) => a - b;
```

```
import('./my-math.js').then(({ sum, sub }) => {
  console.log(sum(1, 2)); // 3
});
```



# ECMAScript 11 - Promise.allSettled

- › *Promise.allSettled* est une nouvelle méthode permettant la combinaison de plusieurs promesses
- › Si parmi ces promesses, une est en erreur, *Promise.all* échoue, *Promise.race* est aléatoire (la plus rapide l'emporte)
- › Avec *Promise.allSettled* on peut intercepter tous les états, succès ou erreur dans le callback de la méthode *then*

```
function timeoutSuccess() {
  return new Promise((resolve, reject) => {
    setTimeout(() => resolve('Data'), Math.random() * 101);
  });
}

function timeoutError() {
  return new Promise((resolve, reject) => {
    setTimeout(() => reject('Error'), Math.random() * 101);
  });
}

Promise.allSettled([timeoutSuccess(), timeoutError()])
  .then((data) => console.log(data));
// [
//   { status: 'fulfilled', value: 'Data' },
//   { status: 'rejected', reason: 'Error' }
// ]
```



**formation.tech**

# ECMAScript Next / ESNext

# ECMAScript Next - Introduction



- ▶ TC39 : Ecma's Technical Committee 39  
Groupe de travail sur la norme JavaScript  
<https://github.com/tc39/ecma262>
- ▶ Membres  
Bloomberg, Microsoft, AirBnb, Apple, Google, Facebook, Mozilla, Meteor, Salesforce, GoDaddy, JS Foundation...
- ▶ 5 étapes :
  - Stage 0: Strawman → Publiée via un formulaire
  - Stage 1: Proposal → A l'étude
  - Stage 2: Draft → Peut encore bouger
  - Stage 3: Candidate → Figée
  - Stage 4: Finished → Sera dans la prochaine norme





# ECMAScript Next - globalThis

- › Dans le navigateur l'objet global s'appelle window, dans Node.js global, parfois this (en mode sloppy)
- › Cette norme prévoit de faire référence à l'objet global via la variable globalThis, quelque soit l'environnement

```
function foo() {  
    globalThis.firstName = 'Romain';  
}  
  
foo();  
console.log(firstName); // Romain
```

# ECMAScript Next - BigInt



- › ESNext prévoit une classe BigInt et une syntaxe littérale pour les déclarer
- › Les opérateurs utilisables sur des nombres le restent sur des BigInt
- › La norme prévoit également des tableaux typés BigInt64Array et BigUint64Array

```
const n = Number.MAX_SAFE_INTEGER;
console.log(n); // 9007199254740991, 2^53 - 1
console.log(n + 1); // 9007199254740992, 2^53
console.log(n + 2); // 9007199254740992, 2^53, same as above

const bigint = 9007199254740991n;
console.log(bigint); // 9007199254740991n, 2^53 - 1
console.log(bigint + 1n); // 9007199254740992n, 2^53
console.log(bigint + 2n); // 9007199254740993n, 2^53 + 1
```

# ECMAScript Next - Class Properties



- ▶ ESNext prévoit que l'on puisse déclarer des propriétés au niveau de la classe

```
class Contact {  
  firstName = 'Romain';  
  hello() {  
    return `Hello my name is ${this.firstName}`;  
  }  
}  
  
const roman = new Contact();  
console.log(roman.hello()); // Hello my name is Romain
```

- ▶ Les propriétés prefixées par un dièse seraient privées

```
class Contact {  
  #firstName = 'Romain';  
  hello() {  
    return `Hello my name is ${this.#firstName}`;  
  }  
}  
  
const roman = new Contact();  
console.log(roman.hello()); // Hello my name is Romain
```

- ▶ React encourage déjà l'utilisation de cette syntaxe dans ses docs

# ECMAScript Next - Optional Chaining



- L'opérateur ? permet de ne pas générer d'erreur lorsque l'on essaye d'accéder à des propriétés sur null ou undefined (comme dans les templates Angular)

```
const contacts = [{  
  firstName: 'Romain',  
  address: {  
    city: 'Paris',  
  },  
}, {  
  firstName: 'Steven',  
}];  
  
for (const contact of contacts) {  
  const city = contact.address?.city;  
  // plutôt que contact.address && contact.address.city  
  console.log('city', city); // Paris, undefined  
}
```



**formation.tech**

# Modules/Loaders/Bundlers JavaScript



**formation.tech**

# Modules JavaScript

# Modules JavaScript - Introduction



## ▶ JavaScript à sa conception

- Objectif : créer des interactions côté client, après chargement de la page
- Exemples de l'époque :
  - Menu en rollover (image ou couleur de fond qui change au survol)
  - Validation de formulaire

## ▶ JavaScript aujourd'hui

- Applications front-end, back-end, en ligne de commande, de bureau, mobiles...
- Applications pouvant contenir plusieurs centaines de milliers de lignes de codes (Front-end de Facebook > 1 000 000 LOC)
- Il faut faciliter le travail collaboratif, en plusieurs fichiers et en limitant les risques de conflit

# Modules JavaScript - Introduction

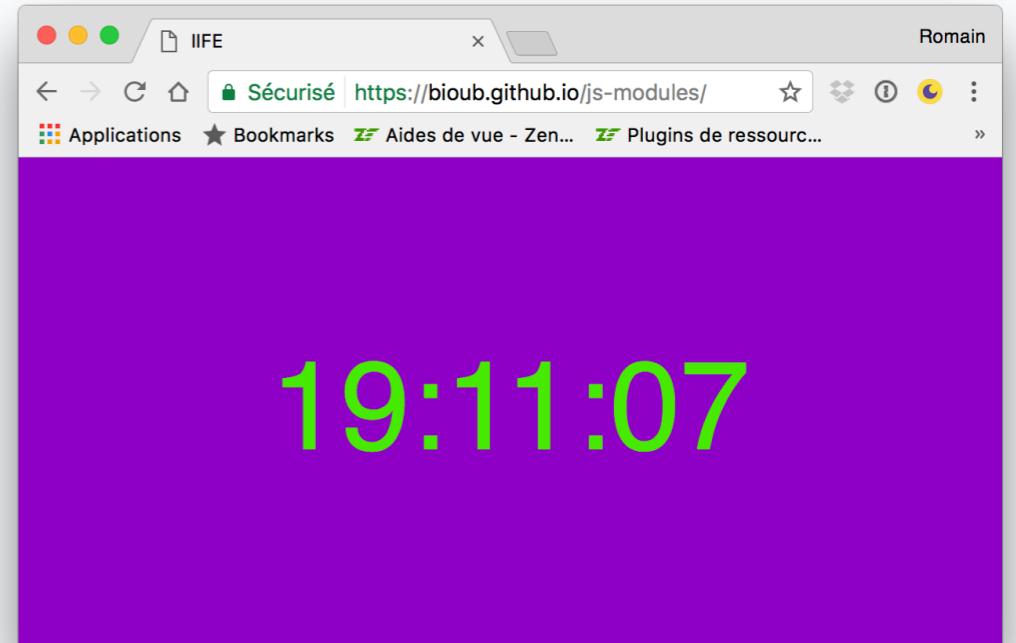
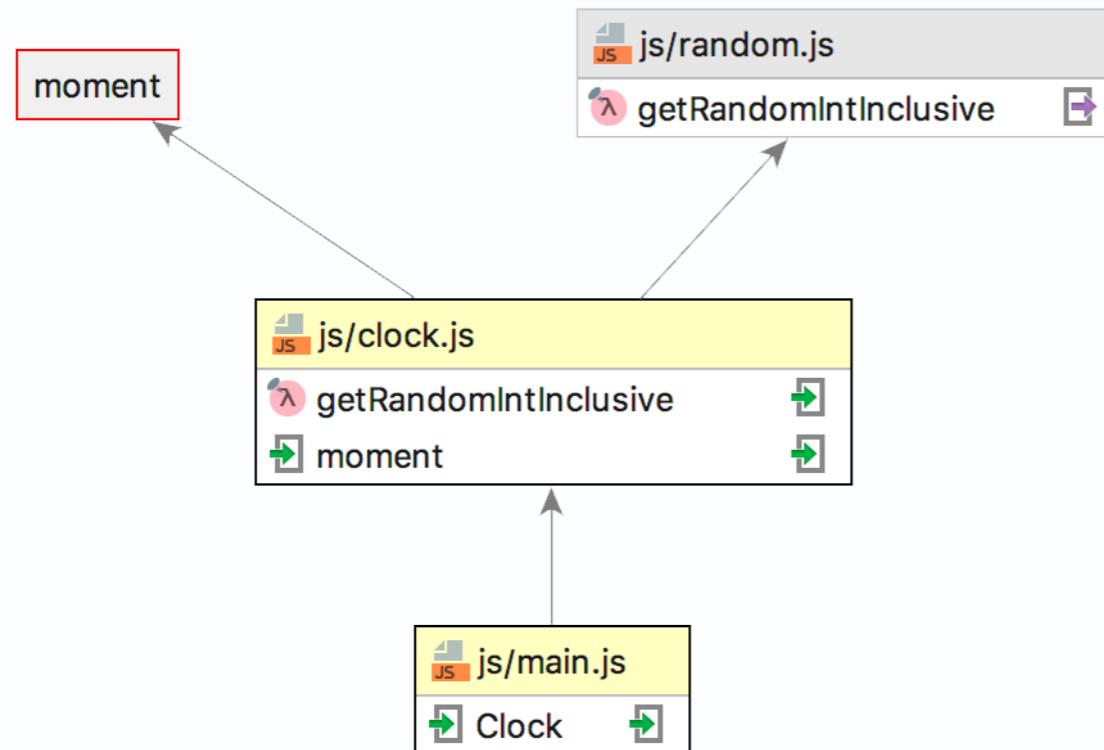


- ▶ Objectifs d'un module JavaScript
  - ▶ Créer une portée au niveau du fichier
  - ▶ Permettre l'export et l'import d'identifiants (variables, fonctions...) entre ces fichiers qui auront désormais leur propre portée
- ▶ Principaux systèmes existants
  - ▶ IIFE / Function Wrapper
  - ▶ CommonJS (fonction synchrone)
  - ▶ AMD
  - ▶ UMD
  - ▶ SystemJS
  - ▶ ES6/ESM (statiques mots clés import / export)
  - ▶ ES2020 : import() (fonction asynchrone)

# Modules JavaScript - Introduction



- ▶ Exemple utilisé pour la suite



- ▶ Le point d'entrée de l'application est le fichier `main.js`, qui dépend de `Clock` défini dans le fichiers `clock.js`, qui dépend lui même de `getRandomIntInclusive` du fichier `random.js` et `moment` définit dans le projet Open Source `moment.js`
- ▶ Exemples : <https://github.com/bioub/js-modules>
- ▶ Démo : <https://bioub.github.io/js-modules/>



# Modules JavaScript - IIFE

## ► Immediately-invoked function expression (IIFE)

```
// random.js
(function (global) {
  'use strict';

  var getRandom = function() {
    return Math.random();
  };

  var getRandomIntInclusive = function(min, max) {
    min = Math.ceil(min);
    max = Math.floor(max);
    return Math.floor(getRandom() * (max - min + 1)) + min;
  };

  global.getRandomIntInclusive = getRandomIntInclusive;
})(this);
```

## ► Une function expression appelée immédiatement

- Limite la portée des identifiants (getRandom, getRandomIntInclusive)
- Permet de renommer localement des dépendances (this → global)



# Modules JavaScript - IIFE

```
// clock.js
(function(global, moment, random) {
  'use strict';

  var Clock = function(parentElt) {
    this.parentElt = parentElt;
  };

  Clock.prototype.update = function() {
    var n = random(0, 255);
    document.body.style.backgroundColor = 'rgb('+n+', '+n+', '+n+')';
    this.parentElt.innerHTML = moment().format('HH:mm:ss');
  };

  Clock.prototype.start = function() {
    this.update();
    setInterval(this.update.bind(this), 1000);
  };
  global.Clock = Clock;
})(this, moment, getRandomIntInclusive);
```

- Pour importer on utilise des variables globales, éventuellement en paramètres d'entrée de la fonction pour pouvoir les renommer localement (getRandomIntInclusion → random)
- Pour exporter on crée des variables globales en étendant l'objet global



# Modules JavaScript - IIFE

## ► Limiter les risques de conflits en IIFE

Exemple : API Maps de Google

```
<!DOCTYPE html>
<html>
<head></head>
<body>
<div id="map"></div>
<script>
  function initMap() {
    var uluru = {lat: -25.363, lng: 131.044};
    var map = new google.maps.Map(document.getElementById('map'), {
      zoom: 4,
      center: uluru
    });
    var marker = new google.maps.Marker({
      position: uluru,
      map: map
    });
  }
</script>
<script async defer src="https://maps.googleapis.com/maps/api/js?callback=initMap"></script>
</body>
</html>
```



# Modules JavaScript - IIFE

## ► Limiter les risques de conflits en IIFE

```
(function(global, google) {
    'use strict';

    var GMap = function (parentElt, options) {
        // ...
    };

    var GMarker = function (parentElt, options) {
        // ...
    };

    global.google = google || {};
    google.maps = google.maps || {};
    google.maps.Map = GMap;
    google.maps.Marker = GMarker;

})(global, google));
```

- Pour limiter les conflits de nom on simule des namespaces.
- Il ne faut créer qu'une seule variable globale du nom de la société, nom du projet... (google dans l'exemple)



# Modules JavaScript - IIFE

## ▶ Utilisation dans le Navigateur

```
<!DOCTYPE html>
<html lang="en">
<head></head>
<body>
  <div class="clock"></div>
  <script src="node_modules/moment/moment.js"></script>
  <script src="js/random.js"></script>
  <script src="js/clock.js"></script>
  <script src="js/main.js"></script>
</body>
</html>
```

- Avec les modules IIFE c'est au développeur de créer les balises script
- Les imports/exports se faisant via des variables globales, il faut maintenir ces balises dans l'ordre des dépendances

## ▶ Utilisation dans Node.js

- Node.js incluant son propre système de module, il n'est pas recommandé d'utiliser des modules IIFE

# Modules JavaScript - CommonJS



## ▶ CommonJS (parfois CJS)

Projet visant à créer des API communs pour du développement JavaScript hors navigateur (console, GUI...)

Exemple : standardiser l'accès aux fichiers

Le projet propose une norme pour le chargement de modules utilisé entre autre par Node.js

<http://wiki.commonjs.org/wiki/Modules/1.1.1>

## ▶ Création d'un module

```
// calculette.js
exports.ajouter = function(nb1, nb2) {
    return Number(nb1) + Number(nb2);
};
```

## ▶ Utilisation

```
// main.js
var calc = require('./calculette');

console.log(calc.ajouter(2, 3)); // 5
```



# Modules JavaScript - CommonJS

## ▶ Export

```
// (function (exports, require, module, __filename, __dirname) {
'use strict';

var getRandom = function() {
    return Math.random();
};

exports.getRandomIntInclusive = function(min, max) {
    min = Math.ceil(min);
    max = Math.floor(max);
    return Math.floor(getRandom() * (max - min + 1)) + min;
};
// });
});
```

- ▶ Le code s'exécute dans une fonction, pas besoin de la créer (ici en commentaire)
- ▶ Cette fonction contient un paramètre exports, qui est un objet que l'on pourra récupérer à l'import (il n'y a plus qu'à l'étendre)



# Modules JavaScript - CommonJS

## ▶ Import

```
// (function (exports, require, module, __filename, __dirname) {  
'use strict';  
  
var moment = require('moment');  
var getRandomIntInclusive = require('./random').getRandomIntInclusive;  
  
var Clock = function(parentElt) {  
  this.parentElt = parentElt;  
};  
  
// ...  
// });
```

## ▶ A l'import on utilise la fonction require()

- ▶ Si le fichier est local on commencera toujours par ./ ou ../
- ▶ Sinon on fait référence à une installation via npm ou bien un fichier se trouvant dans le binaire de Node (fs, http, readline...)
- ▶ La fonction require est synchrone et peut s'utiliser dans un if ou une boucle

# Modules JavaScript - CommonJS



## ▶ Export (autre chose qu'un objet)

```
// (function (exports, require, module, __filename, __dirname) {  
'use strict';  
  
var Clock = function(parentElt) {  
    this.parentElt = parentElt;  
};  
  
// ...  
  
module.exports = Clock;  
  
// });
```

- ▶ Lorsque l'on exporte autre chose qu'un objet, on peut écraser exports en écrivant module.exports (ici on exporte une fonction constructeur)
- ▶ Une bonne pratique pourrait être d'importer en début de fichier et d'exporter toujours en fin de fichier (en pratique c'est rarement le cas)

# Modules JavaScript - CommonJS



- ▶ Utilisation dans le Navigateur
  - Il faut passer par une bibliothèque externe :
    - Soit un bundler comme browserify (historique) ou webpack (moderne)
    - Soit un loader comme SystemJS
  - Les fichiers décrivent eux-mêmes leurs dépendances, il suffit d'indiquer un point d'entrée dans l'application (ou plusieurs)
- ▶ Utilisation dans Node.js
  - Node.js supporte par défaut les modules CommonJS

# Modules JavaScript - AMD



## ▶ Asynchronous Module Definition

CommonJS ne permettant de charger des modules côté client sans transformation préalable. Des développeurs ont imaginé la syntaxe AMD.

## ▶ Fonctionnement

L'utilisation de modules AMD se fait via 2 fonctions globales : require() et define(). define() permet de définir un module et ses dépendances, require définit un point d'entrée dans l'application.



# Modules JavaScript - AMD

## ▶ Export

```
// random.js
define(function() {
  'use strict';

  return {
    getRandom: function () {
      return Math.random();
    },
    // ...
    getRandomIntInclusive: function (min, max) {
      min = Math.ceil(min);
      max = Math.floor(max);
      return Math.floor(this.getRandom() * (max - min + 1)) + min;
    }
  };
});
```

## ▶ define

- ▶ La définition d'un module se passe dans le callback de la fonction define
- ▶ Pour exporter on utilise la valeur de retour
- ▶ Si plusieurs valeurs à exporter il suffit de retourner un tableau ou un objet

# Modules JavaScript - AMD



## ▶ Import

```
// clock.js
define(['moment', './random'], function(moment, random) {
  'use strict';

  var getRandomIntInclusive = random.getRandomIntInclusive;

  var Clock = function(parentElt) {
    this.parentElt = parentElt;
  };

  // ...

  return Clock;
});
```

- ▶ Pour importer on passera un tableau en premier paramètre du define
- ▶ Le callback sera appelé avec les retours des modules importés, dans le même ordre

# Modules JavaScript - AMD



## ▶ Point d'entrée

```
// main.js
require(['./clock'], function(Clock) {
  'use strict';

  var clockElt = document.querySelector('.clock');
  var clock = new Clock(clockElt);
  clock.start();
});
```

## ▶ Require

- ▶ La fonction require permet de définir un point d'entrée dans l'application
- ▶ C'est elle qui va inclure les balises scripts sur la page
- ▶ Idéalement elle ne devrait être utilisée qu'un seul fois par page
- ▶ Les scripts chargent de manière asynchrone et s'exécute lorsqu'il n'ont pas de dépendances ou que toutes leurs dépendances ont été résolues

# Modules JavaScript - AMD



- ▶ Utilisation dans le Navigateur
  - Il faut passer par une bibliothèque externe :
    - Soit un loader comme curl, Require.js ou plus récemment SystemJS
    - Soit un bundler comme webpack qui va faire une transformation
- ▶ Utilisation dans Node.js
  - Non pertinent

# Modules JavaScript - UMD



- ▶ Universal Module Definition (UMD)
  - ▶ Un module universel, qui est à la fois AMD, CommonJS et parfois IIFE  
<https://github.com/umdjs/umd>
  - ▶ Peut s'utiliser avec tous les loaders et les bundlers existants (et sans rien si IIFE)
  - ▶ Pertinent pour les projets où l'on ne peut pas prédire la techno qui sera utilisée pour charger le module
  - ▶ Exemple de module UMD :
    - ▶ jQuery
    - ▶ Lodash → « The Lodash library exported as a UMD module. »  
<https://github.com/lodash/lodash>
    - ▶ Angular → @angular/core/bundles/core.umd.js



# Modules JavaScript - UMD

## ► Exemple de module UMD

```
(function (root, factory) {
  if (typeof exports === 'object') {
    // CommonJS
    module.exports = factory(require('moment'), require('./random'));
  } else if (typeof define === 'function' && define.amd) {
    // AMD
    define(['moment', './random'], function (moment, random) {
      return factory(moment, random);
    });
  } else {
    // IIFE (global var)
    root.Clock = factory(root.moment, root.random);
  }
})(this, function (moment, random) {
  'use strict';

  var getRandomIntInclusive = random.getRandomIntInclusive;

  var Clock = function(parentElt) {
    this.parentElt = parentElt;
  };

  // ...

  return Clock;
}));
```

# Modules JavaScript - UMD



## ▶ Utilisation dans le Navigateur :

- Sans rien si UMD inclus IIFE (pas toujours le cas)
- Avec un bundler : browserify ou webpack
- Avec un loader : curl, Require.js, SystemJS

## ▶ Utilisation dans Node.js

- Avec la fonction require

# Modules JavaScript - ECMAScript Modules



- ES6 introduit la notion de module au niveau du langage (ECMAScript Modules ou ESM)
- Le système est statique, les imports doivent être présent en début de fichier
- Ne permet donc pas de charger dynamiquement le contenu d'un tableau ou d'effectuer un import conditionnel comme avec CommonJS

# Modules JavaScript - ECMAScript Modules



- Exporter
  - Les modules ES permettent d'exporter des valeurs multiples (plutôt que de les regrouper dans un objet)

```
export function getRandom() {
    return Math.random();
}

export function getRandomArbitrary(min, max) {
    return Math.random() * (max - min) + min;
}

export function getRandomInt(min, max) {
    min = Math.ceil(min);
    max = Math.floor(max);
    return Math.floor(Math.random() * (max - min)) + min;
}

export function getRandomIntInclusive(min, max) {
    min = Math.ceil(min);
    max = Math.floor(max);
    return Math.floor(Math.random() * (max - min + 1)) + min;
}
```

# Modules JavaScript - ECMAScript Modules



- On peut également exporter en fin de fichier pour mieux lire les dépendances

```
function getRandom() {
    return Math.random();
}

function getRandomArbitrary(min, max) {
    return Math.random() * (max - min) + min;
}

function getRandomInt(min, max) {
    min = Math.ceil(min);
    max = Math.floor(max);
    return Math.floor(Math.random() * (max - min)) + min;
}

function getRandomIntInclusive(min, max) {
    min = Math.ceil(min);
    max = Math.floor(max);
    return Math.floor(Math.random() * (max - min + 1)) + min;
}

export {
    getRandom,
    getRandomArbitrary,
    getRandomInt,
    getRandomIntInclusive,
}
```

# Modules JavaScript - ECMAScript Modules



- Export par défaut
  - On peut également exporter une valeur par défaut (l'import est particulier)
  - Eviter d'exporter des fonctions multiples dans un objet par défaut

```
export default class Clock {  
    constructor(parentElt) {  
        this.parentElt = parentElt;  
    }  
  
    update() {  
        const r = getRandomIntInclusive(0, 255);  
        const g = getRandomIntInclusive(0, 255);  
        const b = getRandomIntInclusive(0, 255);  
        const now = new Date();  
  
        document.body.style.backgroundColor = `rgb(${r}, ${g}, ${b})`;  
        document.body.style.color = `rgb(${255 - r}, ${255 - g}, ${255 - b})`;  
        this.parentElt.innerHTML = now.toLocaleTimeString();  
    }  
  
    start() {  
        this.update();  
        setInterval(this.update.bind(this), 1000);  
    }  
}
```

# Modules JavaScript - ECMAScript Modules



- On peut renommer les imports (en cas de conflit par exemple)

```
import { format } from 'date-fns/esm';
import { getRandomIntInclusive as rand } from './random';

export default class Clock {
  constructor(parentElt) {
    this.parentElt = parentElt;
  }

  update() {
    const r = rand(0, 255);
    const g = rand(0, 255);
    const b = rand(0, 255);
    const now = new Date();

    document.body.style.backgroundColor = `rgb(${r}, ${g}, ${b})`;
    document.body.style.color = `rgb(${255 - r}, ${255 - g}, ${255 - b})`;
    this.parentElt.innerHTML = now.toLocaleTimeString();
  }

  start() {
    this.update();
    setInterval(this.update.bind(this), 1000);
  }
}
```



# Modules JavaScript - ECMAScript Modules

- ▶ Pour les imports par défaut il faut retirer les accolades
- ▶ L'import peut être renommé directement

```
import Horloge from './clock';

let clockElt = document.querySelector('.clock');
let clock = new Horloge(clockElt);
clock.start();
```

# Modules JavaScript - ECMAScript Modules



## ▶ Utilisation dans le Navigateur :

- Sans rien dans une balise `<script type="module">`
- Avec un bundler : webpack ou Rollup
- Avec un loader : SystemJS

## ▶ Utilisation dans Node.js

- Avec le flag `--experimental-modules`
- Les fichiers doivent avoir l'extension `.jsm`



# Modules JavaScript - Dynamic Imports

- › ESNext prévoit un import dynamique (fonctions asynchrones renvoyant une promesse)
- › webpack est déjà compatible (le code est mis dans un bundle différent, chargé au moment de l'import)

```
document.addEventListener('click', () => {
  import('./calc').then((module) => {
    console.log(module.add(1, 2));
  });
});
```

# Modules JavaScript - Dynamic Imports



- ▶ Utilisation dans le Navigateur :
  - Avec webpack
- ▶ Utilisation dans Node.js
  - Avec webpack



**formation.tech**

# Loaders JavaScript

# Loaders JavaScript - Require.js



- ▶ **Require.js**

Bibliothèque permettant le chargement de modules AMD.

- ▶ **Plugins**

Quelques plugins existent pour charger par exemple des fichiers CSS, des fichiers de traduction, etc...

- ▶ **r.js**

Il est possible d'utiliser un builder pour créer un seul fichier de production, le chargement se ferait sinon en « escalier ».





# Loaders JavaScript - Require.js

## ▶ Chargement en escalier

Name	Status	Type	Initiator	Size	Time	Waterfall	2.00 s	3.00 s	4.00 s	5.00 s	6.00 s	7.00 s	8.00 s	9.00 s	10.00 s	▲1
index-dev.html	200	document	Other	737 B	2.01 s											
normalize.css	200	stylesheet	<a href="#">index-dev.html</a>	8.1 KB	2.42 s											
body.css	200	stylesheet	<a href="#">index-dev.html</a>	473 B	2.04 s											
clock.css	200	stylesheet	<a href="#">index-dev.html</a>	334 B	2.07 s											
require.js	200	script	<a href="#">index-dev.html</a>	84.8 KB	3.97 s											
config.js	200	script	<a href="#">index-dev.html</a>	399 B	2.10 s											
main.js	200	script	<a href="#">index-dev.html</a>	463 B	2.16 s											
clock.js	200	script	<a href="#">require.js:1961</a>	1023 B	2.01 s											
moment.js	200	script	<a href="#">require.js:1961</a>	128 KB	4.60 s											
random.js	200	script	<a href="#">require.js:1961</a>	997 B	2.05 s											

# Loaders JavaScript - SystemJS



## ▶ SystemJS

SystemJS est un loader universel qui sait charger des modules CommonJS, AMD, ES6 et IIFE dans les navigateurs et sous node.js

<https://github.com/systemjs/systemjs>

## ▶ jspm

Afin de faciliter le chargement de modules installés via des gestionnaires de dépendances, jspm permet le chargement de packages npm ou bien de

## ▶ SystemJS Builder

Afin de faciliter le chargement de modules installés via des gestionnaires de dépendance



**formation.tech**

# Bundlers JavaScript

# Bundlers JavaScript - browserify



## ▶ Browserify

Permet de charger des modules CommonJS côté client.

## ▶ Installation :

```
npm install -g browserify
```

## ▶ Transformation en code client :

```
browserify main.js > calculette-browser.js
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title></title>
</head>
<body>
  <script src="calculette-browser.js"></script>
</body>
</html>
```

# Bundlers JavaScript - webpack



## ▶ webpack

webpack est un bundler universel, il sait charger n'importe quel type de modules, AMD, CommonJS, ES6

- ▶ Des loaders supplémentaires permettent de charger des fichiers CSS / LESS / SCSS / i18n / ...
- ▶ Depuis sa version 2, webpack supporte nativement les modules ES6, alors qu'il fallait utiliser un transpileur comme Babel dans la version 1.
- ▶ Le projet open-source le plus financé avec Vue.js (+ 300k\$ par an), son développeur contribue à webpack à temps plein.

# Bundlers JavaScript - webpack



```
const HtmlWebpackPlugin = require('html-webpack-plugin');
const CleanWebpackPlugin = require('clean-webpack-plugin');

module.exports = {
  entry: './src/js/index.js',
  output: {
    path: __dirname + '/dist',
    filename: 'bundle.[hash].js'
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /(node_modules|bower_components)/,
        use: {
          loader: 'babel-loader',
          options: {
            presets: [[ "env", {
              "targets": {
                "browsers": ["> 1% in FR"]
              }
            }]]
          }
        }
      }
    ]
  },
  plugins: [
    new CleanWebpackPlugin(['dist']),
    new HtmlWebpackPlugin({
      template: './src/index.html'
    })
  ]
};
```



**formation.tech**

**npm**



# npm - Gestion de dépendances

- Un gestionnaire de dépendances est un programme qui lance le téléchargement d'une bibliothèque dont dépend votre code, mais également de manière récursive toutes les bibliothèques dont dépendent la bibliothèque installée.
- Equivalent pour du code JavaScript à apt-get
- Principaux gestionnaires de dépendances :
  - Java : Maven, Gradle
  - Ruby : Bundler, gem
  - Python : pip
  - C# : nuget
  - PHP : composer, PEAR
  - Swift / Objective C : CocoaPods
  - JavaScript : npm, yarn, Bower, jspm, pnpm



# npm - Gestion de dépendances

- ▶ Il existe plusieurs programme pour la gestion de dépendances en JavaScript
  - npm  
<https://www.npmjs.com/>  
La norme, s'installe en même temps de Node.js.
  - Yarn  
<https://yarnpkg.com/>  
Développé par Facebook, a une époque considéré comme plus sécurisé et moins exposé au bugs que npm. La version 5 de npm a comblé son retard. Yarn garde l'avantage de paralléliser les téléchargements.
  - pnpm  
<https://pnpm.js.org/>  
Ultra-rapide et moins gourmand en disque, n'installe les dépendances qu'une fois par machine, puis des liens symboliques dans les projets
- ▶ Benchmarks  
<https://github.com/pnpm/node-package-manager-benchmark>



# npm - Gestion de dépendances

- jspm

<https://jspm.org/>

Permet historiquement d'utiliser facilement des modules ES6 avec SystemJS.

Pas recommandé pour de nouveaux projets.

- bower

<https://bower.io/>

Développé par Twitter, historiquement adapté aux dépendances front-end. Pas recommandé pour de nouveaux projets.

## ▶ Annuaires

- Il existe 2 annuaires de dépendances, npm et bower. Yarn, pnpm et jspm utilisent celui de npm.



# npm - Où trouver des bibliothèques ?

## ▶ Sur le registre npm

- Moteur de recherche  
<https://www.npmjs.com>  
<https://npmsearch.com>
- Regarder les stats d'un paquet :  
<https://www.npmjs.com/package/bootstrap>
- Paquets npm les plus utilisées (en nombre de dépendances)  
<https://www.npmjs.com/browse/depended>

## ▶ Sur GitHub

- Recherche par nombre d'étoiles :  
<https://github.com/search?q=stars%3A%3E0>
- Explorer les projets mis en avant par GitHub  
<https://github.com/explore>
- Projets qui reçoivent en ce moment le plus d'étoiles  
<https://github.com/trending>



# npm - Où trouver des bibliothèques ?

- ▶ Awesome Lists

<https://github.com/sindresorhus/awesome>

- Awesome Node.js

<https://github.com/sindresorhus/awesome-nodejs>

- Awesome Frontend

<https://github.com/dypsiLON/frontend-dev-bookmarks>

- Awesome React

<https://github.com/enaqx/awesome-react>

- Awesome Angular

<https://github.com/gdi2290/awesome-angular>

- ▶ Autres

- <https://bestof.js.org/>

- <https://npmcharts.com/>

# npm - Présentation



- Gestionnaire de dépendance de Node.js (en général installé en même temps que Node.js)
- A l'origine, plutôt destiné à du code console ou serveur, bien que des bibliothèques comme jQuery ou Bootstrap y soient présentes depuis toujours
- Aujourd'hui le programme le plus complet mais pas le plus rapide
- Les programmes npm, yarn, pnpm sont incompatibles entre eux, il faut en choisir un dans un projet et s'y tenir





# npm - Le fichier package.json

- Aujourd'hui le fichier package.json est le cœur de la configuration d'un projet JavaScript
- On y retrouve principalement
  - Nos dépendances de prod
  - Nos dépendances de dev
  - Les scripts permettant d'interagir avec le projet (build, serveur de dev, tests...)
  - Des noms, descriptions, auteurs, versions, dépôt git en cas de publication sur npm
  - De la config pour le projet où des outils du projets (linters, tests, builders...)
  - ...

# npm - Le fichier package.json



- Création d'un fichier package.json minimal
- PAS DE COMMENTAIRES DANS UN FICHIER JSON !

```
{}
```

- Pour le créer en ligne de commande
  - Dans un projet (en mode interactif)  
`npm init`
  - Dans un projet (en forçant yes à toutes les questions)  
`npm init -fy`
  - Pour créer un projet depuis un paquet `create-*` (ici un paquet `create-react-app`)  
`npm init react-app mon-app-react`
  - Pour créer un projet en lançant une commande  
`npx create-react-app mon-app-react`



# npm - Installation d'un nouveau paquet

- ▶ Installer un paquet  
`npm install jquery --save`
- ▶ Installer un paquet (alias, --save par défaut depuis npm 5)  
`npm i jquery`
- ▶ Installer un paquet de dev  
`npm i jquery --save-dev`
- ▶ Installer un paquet de dev (alias)  
`npm i jquery -D`
- ▶ Installer une version ancienne d'un paquet  
`npm i jquery@1`  
`npm i jquery@1.12`  
`npm i jquery@1.12.0`
- ▶ Installer ou réinstaller la dernière version d'un paquet (dist-tag latest)  
`npm i jquery@latest`
- ▶ D'autres dist-tags peuvent exister (cf la doc du paquet)  
`npm i jquery@next`



# npm - Lister les dépendances

- ▶ Lister les dépendances

```
npm list
```

```
└ react@16.4.2
  └─ fbjs@0.8.17
    └─ core-js@1.2.7
      └─ isomorphic-fetch@2.2.1
        └─ node-fetch@1.7.3
          └─ encoding@0.1.12
            └─ iconv-lite@0.4.24 deduped
            └─ is-stream@1.1.0
        └─ whatwg-fetch@2.0.4
      └─ loose-envify@1.4.0 deduped
      └─ object-assign@4.1.1 deduped
    └─ promise@7.3.1
    └─ asap@2.0.6 deduped
```

- ▶ deduped signifie que la dépendance est partagée par d'autre (se retrouve à la racine de node\_modules)
- ▶ Lister que nos dépendances directes

```
npm list --depth 0
```

# npm - Déetecter des dépendances à mettre à jour



- ▶ Savoir quoi mettre à jour  
npm outdated
- ▶ 3 cas possibles
  - rouge : mise à jour dispo
  - jaune : migration dispo
  - à jour : n'apparaît pas
- ▶ Versionnage sémantique  
<https://semver.org/>
- ▶ MAJOR.MINOR.PATCH (ex : 1.2.3)
- ▶ New PATCH : pas de changement de comportement (API ou interface identiques)
- ▶ New MINOR : changements rétro-compatibles
- ▶ New MAJOR : changements rétro-incompatibles (lire le guide de migration)

Package	Current	Wanted	Latest	Location
@angular-devkit/build-angular	0.6.3	0.6.8	0.7.5	front-end-scs-angular
@angular/animations	6.0.2	6.0.2	6.1.6	front-end-scs-angular
@angular/cli	6.0.3	6.0.3	6.1.5	front-end-scs-angular
@angular/common	6.0.2	6.0.2	6.1.6	front-end-scs-angular
@angular/compiler	6.0.2	6.0.2	6.1.6	front-end-scs-angular
@angular/compiler-cli	6.0.2	6.0.2	6.1.6	front-end-scs-angular
@angular/core	6.0.2	6.0.2	6.1.6	front-end-scs-angular
@angular/forms	6.0.2	6.0.2	6.1.6	front-end-scs-angular
@angular/http	6.0.2	6.0.2	6.1.6	front-end-scs-angular
@angular/language-service	6.0.2	6.0.2	6.1.6	front-end-scs-angular
@angular/platform-browser	6.0.2	6.0.2	6.1.6	front-end-scs-angular
@angular/platform-browser-dynamic	6.0.2	6.0.2	6.1.6	front-end-scs-angular
@angular/router	6.0.2	6.0.2	6.1.6	front-end-scs-angular
@ng-select/ng-select	2.0.3	2.6.0	2.6.0	front-end-scs-angular
@ngx-translate/core	10.0.1	10.0.2	10.0.2	front-end-scs-angular



# npm - Lockfile

- Le fichier package.json permet de verrouiller
  - Une version majeure : ^1.2.3
  - Une version mineure : ~1.2.3
  - Une version patch : 1.2.3
- Le fichier package-lock.json permet de verrouiller également les dépendances de ce paquet de telle sorte que tous les environnements (postes de dev, prod...) partage les mêmes versions
- Le lockfile permet également de garder l'URL et le checksum de chaque dépendance du projet (perf + sécurité)
- npm maintient un cache sur le disque, les installations ultérieures seront plus rapides
- Les lockfiles de npm, yarn et pnpm sont incompatibles entre eux (parfois même entre 2 versions d'npm !)



# npm - Mettre à jour

- Mettre à jour tous les paquets packages installés  
`npm update`
- Mettre à jour un seul paquet  
`npm update jquery`
- Migrer un paquet vers une version majeure  
`npm i jquery@2`
- Migrer un paquet vers la dernière version majeure  
`npm i jquery@latest`
- Désinstaller  
`npm uninstall lodash`



# npm - Lancer un script

- ▶ Lancer un script

```
npm run-script nom-du-script
```

- ▶ Lancer un script (alias)

```
npm run nom-du-script
```

- ▶ Certains script ont des alias (start, test...)

```
npm run-script test
```

```
npm run test
```

```
npm test
```

```
npm t
```

- ▶ Certains scripts peuvent s'exécuter automatiquement  
postinstall, prepublish, pretest...

- ▶ Lancer un script permet d'exécuter un programme local



# npm - Configuration

- ▶ Utilisation d'un proxy

`npm config set proxy http://host:8080`

`npm config set proxy http://user:pass@host:8080`

- ▶ Supprimer une config

`npm config rm proxy`

- ▶ Lister les configs

`npm config list`



**formation.tech**

# Node.js

# Node.js - Introduction



- Crée 2009 par Ryan Dahl
  - A l'origine, Ryan Dahl voulait simplifier la création d'une barre d'upload.
- Sponsorisé par la société Joyent.
- Un programme en ligne de commande combinant :
  - le moteur JavaScript V8 de Chrome
  - une boucle d'événement
  - une gestion bas niveau des entrées/sorties
- Un système en production :
  - Chez des startups à la pointe : Airbnb, ...
  - Dans des grands groupes : Microsoft, PayPal, Walmart, Linkedin



# Node.js - Installation

- ▶ Windows  
Exécutables : <https://nodejs.org/download/>
- ▶ OS X  
Exécutables : <https://nodejs.org/download/>  
Ou via homebrew : brew install node
- ▶ Debian / Ubuntu  
sudo apt-get update  
sudo apt-get install nodejs npm
- ▶ Pensez à ajouter le répertoire de Node au Path.



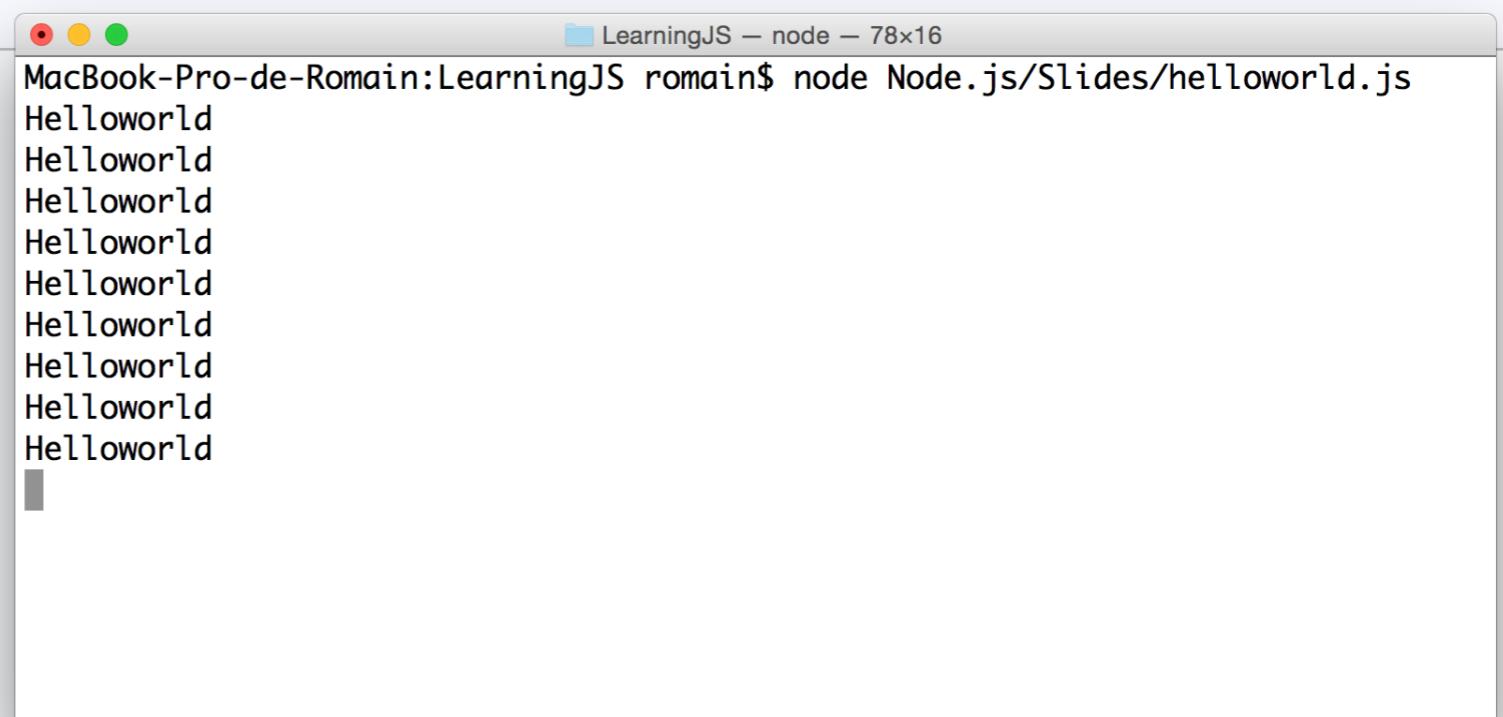
# Node.js - Helloworld

- ▶ Lancement du programme  
node FILE\_PATH[.js]

```
/* Un simple helloworld */

/** @function helloworld */
function helloworld() {
  'use strict'; // bonne pratique
  console.log('Helloworld');
}

setInterval(helloworld, 1000);
```



A screenshot of a terminal window titled "LearningJS — node — 78x16". The window shows the command "MacBook-Pro-de-Romain:LearningJS roman\$ node Node.js/Slides/helloworld.js" followed by nine lines of the word "Helloworld" printed sequentially.

```
MacBook-Pro-de-Romain:LearningJS roman$ node Node.js/Slides/helloworld.js
Helloworld
Helloworld
Helloworld
Helloworld
Helloworld
Helloworld
Helloworld
Helloworld
Helloworld
```



# Node.js - Débogage

- En ligne de commande  
`node inspect FILE_PATH[.js]`
- Avec les DevTools de Chrome  
`node --inspect-brk FILE_PATH[.js]`  
Puis sous Chrome aller à l'URL <chrome://inspect>
- Avec les IDEs Webstorm ou Visual Studio Code



# Node.js - 2 utilisations

- ▶ Pour les programmes en ligne de commande, ex :
  - build: webpack / grunt / gulp...
  - linters: eslint / tslint / stylelint...
  - tests: mocha / jest / karma
  - serveurs web de dev: http-server / live-server...
  - ...
- ▶ Pour les programmes serveur
  - Applications traditionnelles, HTML rendu côté serveur
  - API REST
  - Serveur WebSocket
  - ...

# Node.js - Pourquoi JavaScript côté serveur ?



```
const http = require('http');

http.createServer((req, res) => {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello, world !');
}).listen(8080);
```

- ▶ Avantage du JavaScript côté serveur
  - Performance : le côté non-bloquant de JavaScript le rend particulièrement performant, plus besoin de gérer les problèmes inter-thread ou inter-processus
  - Réutilisation : une bibliothèque ou un composant peut être utilisé sur le client comme sur le serveur
  - Apprentissage : vous connaissez déjà JavaScript
  - Ecosystème : le nombre de bibliothèques open-source (langage le plus populaire sur GitHub)

# Node.js - Event Loop



```
const http = require('http');

http.createServer((req, res) => {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello, world !');
}).listen(8080);
```

- Node.js implémente côté C++ une boucle d'événement et une gestion non-bloquante des entrées/sorties
- Ici lorsque qu'un requête HTTP arrive sur le port 8080 la fonction de rappel passée en argument de createServer est appelée
- Il faut quelques millisecondes pour exécuter ce callback, le reste du temps JavaScript est libre de faire autre chose (exécuter des requêtes concurrentes, se connecter à une base de données, écrire dans un fichier...)

# Node.js - Documentation



- ▶ Guides  
<https://nodejs.org/en/docs/guides/>
- ▶ API  
<https://nodejs.org/api/>
- ▶ 3 niveaux de "stabilité"

Stability: 0 - Deprecated. This feature is known to be problematic, and changes may be planned. Do not rely on it. Use of the feature may cause warnings to be emitted. Backwards compatibility across major versions should not be expected.

Stability: 1 - Experimental. This feature is still under active development and subject to non-backwards compatible changes, or even removal, in any future version. Use of the feature is not recommended in production environments. Experimental features are not subject to the Node.js Semantic Versioning model.

Stability: 2 - Stable. The API has proven satisfactory. Compatibility with the npm ecosystem is a high priority, and will not be broken unless absolutely necessary.

# Node.js - API



- Assertion Testing
- Async Hooks
- Buffer
- C++ Addons
- C/C++ Addons - N-API
- Child Processes
- Cluster
- Command Line Options
- Console
- Crypto
- Debugger
- Deprecated APIs
- DNS
- Domain
- ECMAScript Modules
- Errors
- Events
- File System
- Globals
- HTTP
- HTTP/2
- HTTPS
- Inspector
- Internationalization
- Modules
- Net
- OS
- Path
- Performance Hooks
- Process
- Punycode
- Query Strings
- Readline
- REPL
- Stream
- String Decoder
- Timers
- TLS/SSL
- Trace Events
- TTY
- UDP/Datagram
- URL
- Utilities
- V8
- VM
- Worker Threads
- ZLIB

# Node.js - API



- ▶ Ne sont pas (ou pas complètement) des APIs
  - C++ Addons et C/C++ Addons - N-API  
Permet de compiler du code C ou C++ et de le "binder" (le rendre accessible) au JavaScript de Node.js
  - Command Line Options  
Arguments du programme Node
  - Debugger  
Un debugger en ligne de commande
  - Deprecated APIs  
La liste des APIs dépréciés
  - ECMAScript Modules  
L'implémentation des Modules ECMAScript (ESM ou ES6 Modules)
  - Internationalization  
Comment compiler Node.js pour améliorer le support i18n
  - REPL  
Permet de lancer Node en mode interactif et de saisir du code dans la console

# Node.js - API



- ▶ Les APIs dépréciés (seront supprimé à l'avenir)
  - Domain
  - Punycode
  - <https://nodejs.org/dist/latest/docs/api/deprecations.html>
- ▶ Les APIs expérimentaux (leur implémentation peut évoluer)
  - Async Hooks
  - ECMAScript Modules
  - HTTP/2
  - Inspector
  - Performance Hooks
  - Trace Events
  - Worker Threads



# Node.js - Console

- Le module console (global) permet de logger dans la console et de réaliser des benchmarks

```
console.time('100-elements');
for (var i = 0; i < 100; i++) {
  console.log(i);
}
console.timeEnd('100-elements');
// 100-elements: 8ms
```

# Node.js - Timers



- ▶ Le module Timers (global) contient les fonctions pour différer l'exécution de callbacks.

```
setTimeout(function() {
  console.log('1 fois dans 3 secondes');
}, 3000);

var intervalId = setInterval(function() {
  console.log('toutes les 2 secondes');
}, 2000);

setTimeout(function() {
  console.log('Bye bye');
  clearInterval(intervalId);
}, 15000);
```

# Node.js - File System



- ▶ Le module File System (`require('fs')`) permet les accès aux disques, fichiers, dossiers, droits, etc...

```
var fs = require('fs');

try {
  var stats = fs.statSync('./dist');
}
catch(e) {
  fs.mkdirSync('./dist');
}

var fichiers = fs.readdirSync('./src');

for (var i=0; i<fichiers.length; i++) {
  var fichier = fichiers[i];
  var src = './src/' + fichier;
  var dest = './dist/' + fichier;

  var contenu = fs.readFileSync(src);
  fs.writeFileSync(dest, contenu);
}
```

# Node.js - Net



- Le module net (require(net)) permet les accès réseau

```
var net = require('net');
var server = net.createServer(function(c) { //'connection' listener
  console.log('client connected');
  c.on('end', function() {
    console.log('client disconnected');
  });
  c.write('hello\r\n');
  c.pipe(c);
});
server.listen(8124, function() { //'listening' listener
  console.log('server bound');
});
```



# Node.js - Serveur Net

## ► Un chat serveur avec Net

```
var net = require('net');

var clients = {}, cpt = 0;

var server = net.createServer(function(c) { // 'connection' listener
    var me = 'c' + (++cpt);
    console.log('client connected');

    clients[me] = c;
    c.on('end', function() {
        //clients[me].end();
        delete clients[me];
    });
    c.write('Bienvenue sur le Chat !!! (telnet : taper exit pour quitter)\r\n');

    c.on('data', function(chunk) {
        for (var cid in clients) {
            if (clients.hasOwnProperty(cid)) {
                if (chunk.toString().indexOf('exit') === 0) {
                    clients[me].end();
                    delete clients[me];
                    break;
                }

                if (cid !== me) {
                    clients[cid].write(chunk.toString());
                }
            }
        }
    })
});

server.listen(8124, function() { // 'listening' listener
    console.log('server bound');
});
```



# Node.js - Client Net

## ► Un chat client avec Net

```
var net = require('net');
var readline = require('readline');

var rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

rl.question("Quel est ton pseudo ? ", function(pseudo) {
  console.log("Bienvenue sur le chat", pseudo);

  var client = net.connect({port: 8124}, function() { //connect' listener
    console.log('(connecté au serveur)');
    process.stdin.on('readable', function() {
      var chunk = process.stdin.read();
      if (chunk !== null) {
        var msg = chunk.toString();
        msg = msg.substr(0, msg.length - 1); // on retire le \n
        client.write(pseudo + ': ' + msg);
      }
    });
    client.on('data', function(data) {
      console.log(data.toString());
      //client.end();
    });
    client.on('end', function() {
      console.log('disconnected from server');
    });
  });
  rl.close();
});
```

# Node.js - HTTP



- ▶ CreateServer

Contrairement à d'autres technologies, l'implémentation du serveur HTTP se fait dans l'application.

- ▶ Callback

Une fonction de rappel est associée au serveur. Elle sera appelée à chaque requête HTTP.

- ▶ Objets Request et Response

Node.js abstrait la requête (IncomingMessage) et la réponse (ServerResponse), le callback doit créer une réponse valide avant la fin de son exécution.

```
var http = require('http');

http.createServer(function(req, res) {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello, world !');
}).listen(8080);
```



# Node.js - HTTPS

## ▶ HTTPS

Le serveur HTTPS démarre avec une clé privée et un certificat. Les serveurs HTTP et HTTPS cohabitent dans la même application.

```
var https = require('https');
var http = require('http');
var fs = require('fs');

function serveur(req, res) {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello, world !');
}

var options = {
    key: fs.readFileSync('./key.pem', 'utf8'),
    cert: fs.readFileSync('./server.crt', 'utf8')
};

http.createServer(serveur).listen(80);
https.createServer(options, serveur).listen(443);
```

# Node.js - Gestion des routes



- ▶ Pages vs Serveur

Node.js possède une implémentation HTTP très bas niveau. Quand en Java ou en PHP un fichier équivaut à une URL, Node.js lui est le serveur dans son ensemble.

- ▶ Routes

La gestion des URL se fait donc en internes, l'association entre un chemin d'accès (Méthode HTTP + URL) et du code s'appelle une route.

- ▶ Router

L'ensemble des routes est configurée dans un composant que l'ont appelle un router.



# Node.js - Gestion des routes

- ▶ Implémentation d'un router basique

Dans l'exemple ci-dessous, on implémente un router à l'aide d'un simple switch.

```
const http = require('http');

const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');

    switch (req.url) {
        case '/':
            res.end('Hello World');
            break;
        case '/toto':
            res.end('Hello Toto');
            break;
        default:
            res.statusCode = 404;
            res.end('404 Not Found');
            break;
    }
});

server.listen(8080, () => {
    console.log('Server started http://localhost:8080');
});
```

# Node.js - Exercice



- Créer un serveur web contenant 2 routes :  
/contacts et /contacts/123
- Sur la route /contacts retourner un tableau de contact (prenom + nom)
- Sur la route /contacts/123 retourner le contact dont l'id est 123 (utiliser la fonction find du type Array)
- Remplacer le 123 dans la route /contacts/123 pour permettre de passer un id quelconque (utiliser une expression régulière ou bien la méthode substr du type String)



**formation.tech**

# Express



# Express - Framework Web

- ▶ Définition d'un framework web :  
Ensemble de composants logiciels permettant d'architecturer un projet logiciel.
- ▶ Différences par rapport à une bibliothèque :  
Le framework ne se destine pas à une tache précise (ensemble de bibliothèques)  
Le framework instaure un cadre de travail (squelettes d'application, documentation sur l'architecture...)



# Express - Frameworks web connus

- ▶ Java  
Struts (2000), Spring (2003), GWT (2006), Play (2007)...
- ▶ Ruby  
Ruby on Rails (2005), Sinatra (2007)...
- ▶ Python  
Django (2005)...
- ▶ PHP  
Symfony, Zend Framework, CakePHP, CodeIgniter...



# Express - Frameworks web en JS

- Clients
  - AngularJS (2010), Ember.js (2011)
- Server
  - Express (2009), Hapi (2012)
- Fullstack (Client + Server)
  - Meteor (2012), Sails.js (2012)...



# Express - Introduction

- ▶ Express
  - Framework pour Node.js le plus populaire, créé en 2009, aujourd'hui en version 4.
  - Permet d'architecturer plus facilement le serveur web.
  - Très souvent utilisé pour construire des APIs REST.
- ▶ Avantages sur le module HTTP de Node.js
  - Gestion des URLs et des méthodes HTTP
  - Approche MVC
  - Utilisation de middlewares qui permettent d'étendre le code
  - De nombreux middleware open-source existent
  - Construit comme une surcouche de HTTP, les objets Request et Response sont simplement étendus
- ▶ Installation
  - `npm install express --save`



# Express - Helloworld

```
var express = require('express');

var app = express();

app.get('/', function(req, res) {
  res.send('Hello world');
});

app.listen(8080);

console.log("URL http://localhost:8080/");
```

localhost:8080

localhost:8080

Hello world

Elements Network Sources Timeline Profiles Resources Audits Console

Name

localhost

Headers Preview Response Timing

General

- Remote Address: [::1]:8080
- Request URL: http://localhost:8080/
- Request Method: GET
- Status Code: 200 OK

Response Headers

- Connection: keep-alive
- Content-Length: 11
- Content-Type: text/html; charset=utf-8
- Date: Fri, 23 Oct 2015 16:31:47 GMT
- ETag: W/"b-8bd69e52"
- X-Powered-By: Express

1 requests | 196 B transferred | Finish...

# Express - MVC

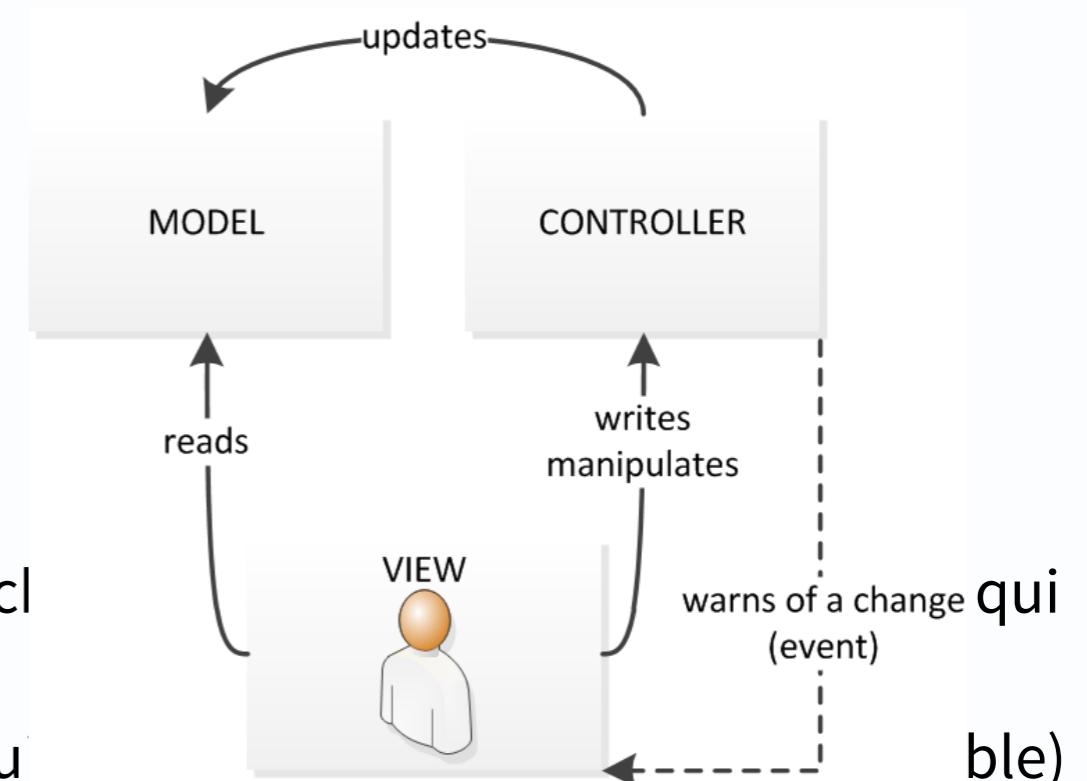


- ▶ Définition  
L'architecture MVC est un Design Pattern apparu en Smalltalk et très répandu dans les frameworks web
- ▶ Objectif  
L'objectif est de séparer les responsabilités de 3 types de composants : le Modèle (Model), la Vue (View), le Contrôleur (Controller)
- ▶ Documentation :  
<http://martinfowler.com/eaaCatalog/modelViewController.html>  
<http://martinfowler.com/eaaDev/uiArchs.html>  
<http://fr.wikipedia.org/wiki/Modèle-vue-contrôleur>



# Express - MVC

- ▶ Modèle  
Données, accès aux données, validation
- ▶ Vue  
Rendu. Se limiter à :
  - affichage de variable
  - bloc conditionnels if .. else if .. else (ex : affich dépend d'une authentification)
  - boucles foreach (uniquement foreach, ce qu
  - appel à des fonctions de filtrage, de formatage, de rendu (parfois appelées aides de vues)
- ▶ Contrôleur  
Analyse de la requête, interrogation du Modèle, transmission des données à la vue, gestion des erreurs, des redirections...



# Express - Middleware



- ▶ Définition  
Un middleware est une fonction qui va s'exécuter en amont ou en aval d'une requête dans Express pour l'étendre simplement.
- ▶ Exemple  
Logs de requêtes, authentification, gestion des requêtes Cross-Domaines, support d'un moteur de templates...
- ▶ Connect  
Historiquement Express utilisait le module npm connect pour la mise en place de middleware. A partir d'Express 4, les développeurs d'Express ont développé leur propre système de middleware tout en gardant la compatibilité avec Connect.

# Express - Express Generator



- ▶ Introduction

Express fournit un générateur de squelette d'application pour démarrer rapidement ses projets web (plutôt adapté aux rendus HTML)

- ▶ Installation

```
npm install -g express-generator
```

- ▶ Création du squelette d'application

```
express Helloworld
```

- ▶ Installation des dépendances

```
cd Helloworld && npm install
```

- ▶ Lancement de l'application

```
DEBUG=HelloworldEJS:* npm start
```



# Express - Express Generator

- ▶ Autres options d'installation du squelette

```
MacBook-Pro-de-Romain:Helloworld romain$ express -h

Usage: express [options] [dir]

Options:

-h, --help          output usage information
-V, --version       output the version number
-e, --ejs           add ejs engine support (defaults to jade)
--hbs              add handlebars engine support
-H, --hogan         add hogan.js engine support
-c, --css <engine> add stylesheet <engine> support (less|stylus|compass|sass) (defaults to plain css)
--git              add .gitignore
-f, --force         force on non-empty directory

MacBook-Pro-de-Romain:Helloworld romain$
```

- ▶ Choix du moteur de templates  
Jade, EJS, Handlebars, Hogan.js
- ▶ Choix d'un préprocesseur CSS  
CSS, Less, Stylus, Compass, Sass

# Express - Express Generator



- **app.js**  
Configuration de l'application, objet principal
- **bin/www**  
Démarrage du serveur
- **package.json**  
Dépendances npm
- **public**  
Fichiers statiques (images, scripts client, css, pdf...)
- **routes**  
Contrôleurs, configuration des URLs
- **views**  
Fichiers de rendus (ici au format Jade)

```
├── app.js
└── bin
    └── www
├── node_modules
└── ...
├── package.json
└── public
    ├── images
    ├── javascripts
    └── stylesheets
        └── style.css
├── routes
    ├── index.js
    └── users.js
└── views
    ├── error.jade
    ├── index.jade
    └── layout.jade
```

# Express - Express Generator



## ▶ bin/www

- Dépendances
- Définition du port (variable d'environnement)
- Création du serveur
- Démarrage du serveur
- Listeners sur erreurs et démarrage

```
#!/usr/bin/env node

/**
 * Module dependencies.
 */

var app = require('../app');
var debug = require('debug')('Helloworld:server');
var http = require('http');

/**
 * Get port from environment and store in Express.
 */
var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

/**
 * Create HTTP server.
 */

var server = http.createServer(app);

/**
 * Listen on provided port, on all network interfaces.
 */

server.listen(port);
server.on('error', onError);
server.on('listening', onListening);

// ...
```



# Express - Express Generator

## ▸ app.js

- Dépendances
- Chargement des routes
- Création de l'objet app
- Définition du moteur de template
- Définition des middlewares à app
- Définition des contrôleurs sur un module
- Middleware pour les erreurs 404
- Middleware pour afficher les erreurs

```
var express = require('express');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');

var routes = require('./routes/index');
var users = require('./routes/users');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');

app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', routes);
app.use('/users', users);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
  next(err);
});

// error handlers
// ...
```

# Express - Express Generator



- ▶ routes/index.js
  - Dépendances
  - Association d'un contrôleur à la l'URL GET /
  - Appel de la vues en transmettant la variable title (contenu 'Express')

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next)
{
    res.render('index', { title:
    'Express' });
});

module.exports = router;
```

# Express - Express Generator



## › views/index.jade

- Jade : Syntaxe très concise, l'indentation fait l'imbrication des balises, parenthèses pour les attributs
- Héritage du layout
- Remplacement du block content du layout par celui de la vue (inspiré de Django Template Engine, Twig...)
- Création de la vue

```
// views/index.jade
extends layout
block content
  h1= title
  p Welcome to #{title}
```

```
// views/layout.jade
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    block content
```

# Express - Express Generator



- ▶ views/index.ejs
  - Syntaxe plus simple que Jade proche de PHP, ASP, JSP
  - <%= title %> : écriture de la variable title

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <p>Welcome to <%= title %></p>
  </body>
</html>
```

# Express - Routeur



- ▶ Réponse à toutes les méthodes HTTP

```
router.all('/api/*', requireAuthentication);
```

- ▶ Réponse aux requêtes sur certaines méthodes HTTP

Méthodes HTTP : get, post, put, head, delete, options, trace, copy, lock, mkcol, move, purge, propfind, proppatch, unlock, report, mkactivity, checkout, merge, m-search, notify, subscribe, unsubscribe, patch, search, connect

```
router.get('/', function(req, res){  
    res.send('hello world');  
});
```

- ▶ Avec une RegExp

```
router.get(/^\/commits\/((\w+)(?:\.\.(\w+))?)$/, function(req, res){  
    var from = req.params[0];  
    var to = req.params[1] || 'HEAD';  
    res.send('commit range ' + from + '...' + to);  
});
```



# Express - Routeur

- ▶ Route avec paramètres nommés

```
router.get('/:id', function(req, res, next) {  
  var id = req.params.id;  
  
  if (!model[id-1]) {  
    return next();  
  }  
  
  res.json({  
    data: model[id-1]  
  });  
});
```

- ▶ Ne pas confondre avec la query string  
Ex : /contacts?page=1&limit=100

```
// GET /search?q=tobi+ferret  
req.query.q  
// => "tobi ferret"
```



# Express - Middleware

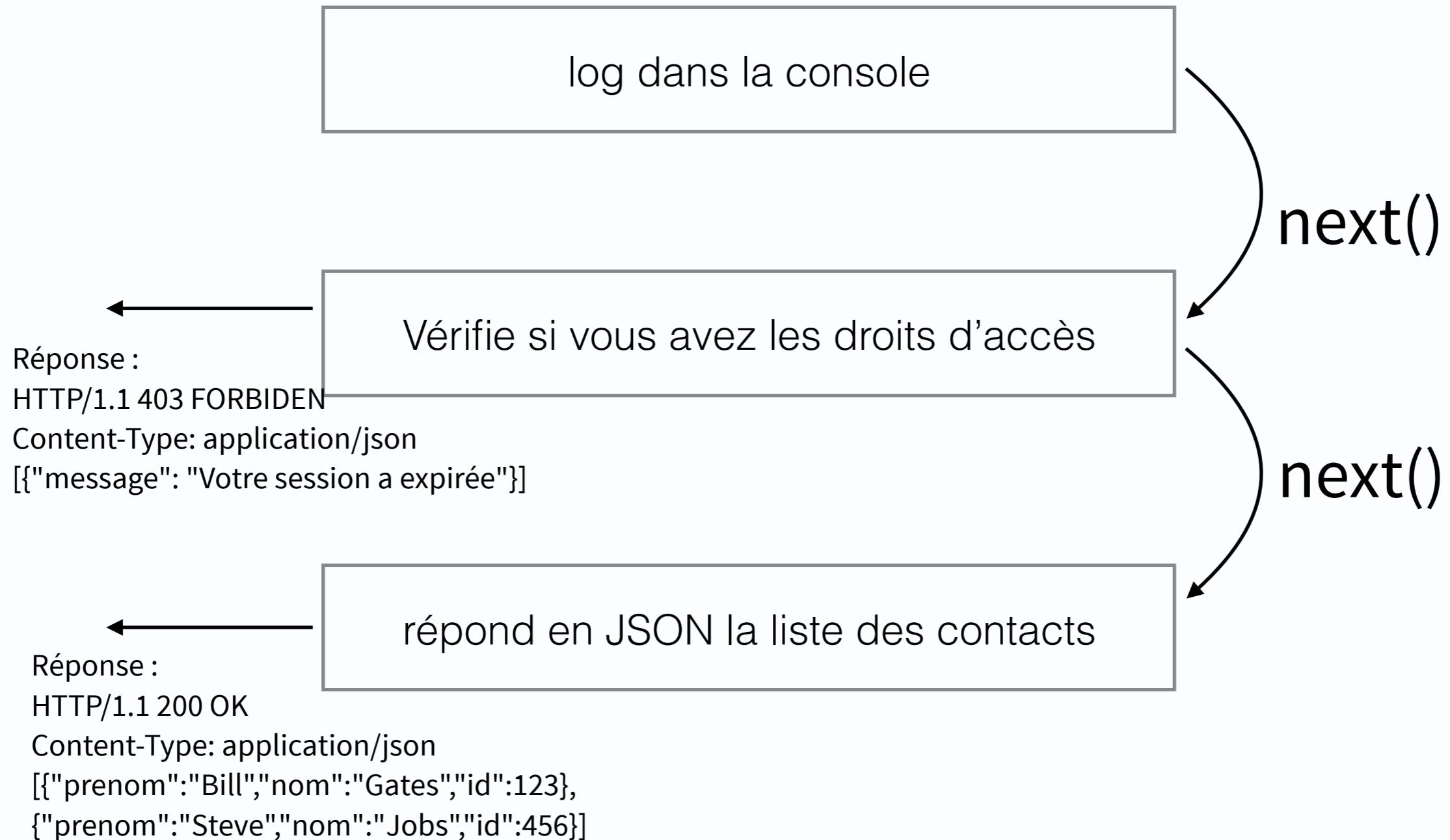
- ▶ Middleware  
Fonction qui s'exécute en amont ou en aval d'un contrôleur
- ▶ Exemple  
Ajoute les entêtes à la réponse HTTP permettant d'autoriser les requêtes Cross-Domain

```
router.use(function(req, res, next) {  
    res.setHeader('Access-Control-Allow-Origin', '*');  
    res.setHeader('Access-Control-Allow-Headers', 'X-Requested-With');  
    next();  
});
```



# Express - Middleware

## Requête : GET / HTTP/1.1



# Express - Middleware



- ▶ 3 middlewares
  1. Router qui contient toutes les URLs préfixées par /users
  2. Middleware appelé si l'un des contrôleurs ou middlewares précédents appelle next(), permet de gérer simplement les erreurs 404
  3. Middleware appelé si l'un des contrôleurs ou middlewares précédents appelle next(err) ou les erreurs non-interceptées

```
app.use('/users', users);

app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
  next(err);
});

app.use(function(err, req, res, next) {
  res.status(err.status || 500);
  res.render('error', {
    message: err.message,
    error: err
  });
});
```

# Express - Middleware



## ▶ Request

L'objet Request hérite de IncomingMessage du module HTTP

## ▶ Middleware body-parser

Le middleware body-parser ajouter la propriété body à l'objet request avec le contenu du corps de requête parseé

```
var express = require('express');
var bodyParser = require('body-parser');

var app = express();
app.use(bodyParser.urlencoded({ extended: false }));

app.get('/', function(req, res) {
    var html = '<form method="post">';
    html += '  <p>Prénom : <input name="prenom"></p>';
    html += '  <p>Nom : <input name="nom"></p>';
    html += '  <p><button type="submit">Valider</button></p>';
    html += '</form>';

    res.send(html);
})

app.post('/', function(req, res) {
    // Prénom : Romain, nom : Bohdanowicz
    res.send(`Prénom : ${req.body.prenom}, nom : ${req.body.nom}`);
})

app.listen(3000);
```

# Express - Middleware



## ▶ Middleware multer

Le middleware multer ajouter la propriété file ou files (upload multiple) à l'objet request et contient des informations sur le fichier uploadé.

```
var express = require('express');
var multer = require('multer');

var app = express();
var upload = multer({ dest: 'uploads/' });

app.get('/', function(req, res) {
  var html = '<form method="post" enctype="multipart/form-data">';
  html += '  <p>Fichier : <input type="file" name="fichier"></p>';
  html += '  <p><button type="submit">Valider</button></p>';
  html += '</form>';

  res.send(html);
});

app.post('/', upload.single('fichier'), function(req, res){
  console.log(req.file);
  res.status(204).end();
});

app.listen(3000);
```

```
{ filename: 'fichier',
  originalname: '2010_Q3.pdf',
  encoding: '7bit',
  mimetype: 'application/pdf',
  destination: 'uploads/',
  filename: '799e08c05ef96ac6ec6ac5b714941161',
  path: 'uploads/799e08c05ef96ac6ec6ac5b714941161',
  size: 80108 }
```



# Express - JSON

## ▶ JSON

L'objet Response contient une méthode json qui sérialise un objet et renvoie les bons entêtes HTTP. Associés au méthodes de l'objet Request et aux middleware body-parser et cors, Express est le framework idéal pour la mise en place d'un API REST qui communique en JSON.

```
var app = express();
app.use(cors({ allowedHeaders: 'X-Requested-With' }));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));

app.use('/api/v1/contacts', apiContacts);

app.listen(80);
```



# Express - API REST

```
var express = require('express');
var contacts = require('../model/contacts').slice(0);

var api = express.Router();

api.get('/', function(req, res) {
    res.json({contacts});
});

api.get('/:id', function(req, res, next) {
    var id = parseInt(req.params.id);
    var contact = contacts.find(elt => elt.id === id);

    if (!contact) return next();

    res.json({contact});
});

api.post('/', function(req, res, next) {
    var contact = req.body;

    contact.id = contacts[contacts.length-1].id + 1;
    contacts.push(contact);

    res.status(201);
    res.json(contact);
});

module.exports = api;
```



# Express - Designer un API REST

- Guide inspiré de Google, Facebook, Twitter, GitHub...  
<http://blog.octo.com/designer-une-api-rest/>
- Guide inspiré par Heroku  
<https://github.com/interagent/http-api-design>
- Richardson Maturity Model  
<https://martinfowler.com/articles/richardsonMaturityModel.html>

# Express - Exercice



- ▶ Créer un API RESTful avec Express permettant d'interagir avec un tableau de contacts en JSON
- ▶ 5 routes :
  - GET /api/contacts - Lister les contacts
  - GET /api/contacts/:id - Afficher un contact
  - POST /api/contacts - Ajouter un contact
  - PUT /api/contacts/:id - Modifier un contact
  - DELETE /api/contacts/:id - Supprimer un contact



**formation.tech**

# NoSQL

# NoSQL - Introduction



## ▶ NoSQL

Not Only SQL, le nom qu'on donne au mouvement depuis quelques années de ne pas tout stocker sous la forme de base de données relationnels (MySQL, SQLite, PostgreSQL, Oracle, SQL Server...).

A l'origine en 2009, le nom donné à un meetup à San Francisco.

Parfois appelé « NoRel » (« not only relational ») pour ne pas prêter à confusion.

## ▶ Intérêts

Performance, scalabilité, haute-disponibilité

## ▶ Catégories

- Clé / valeur (Redis / Memcached...)
- Orienté Colonne (HBase / Cassandra...)
- Orienté Document (MongoDB / CouchDB...)
- Orienté Graphe (Neo4j)



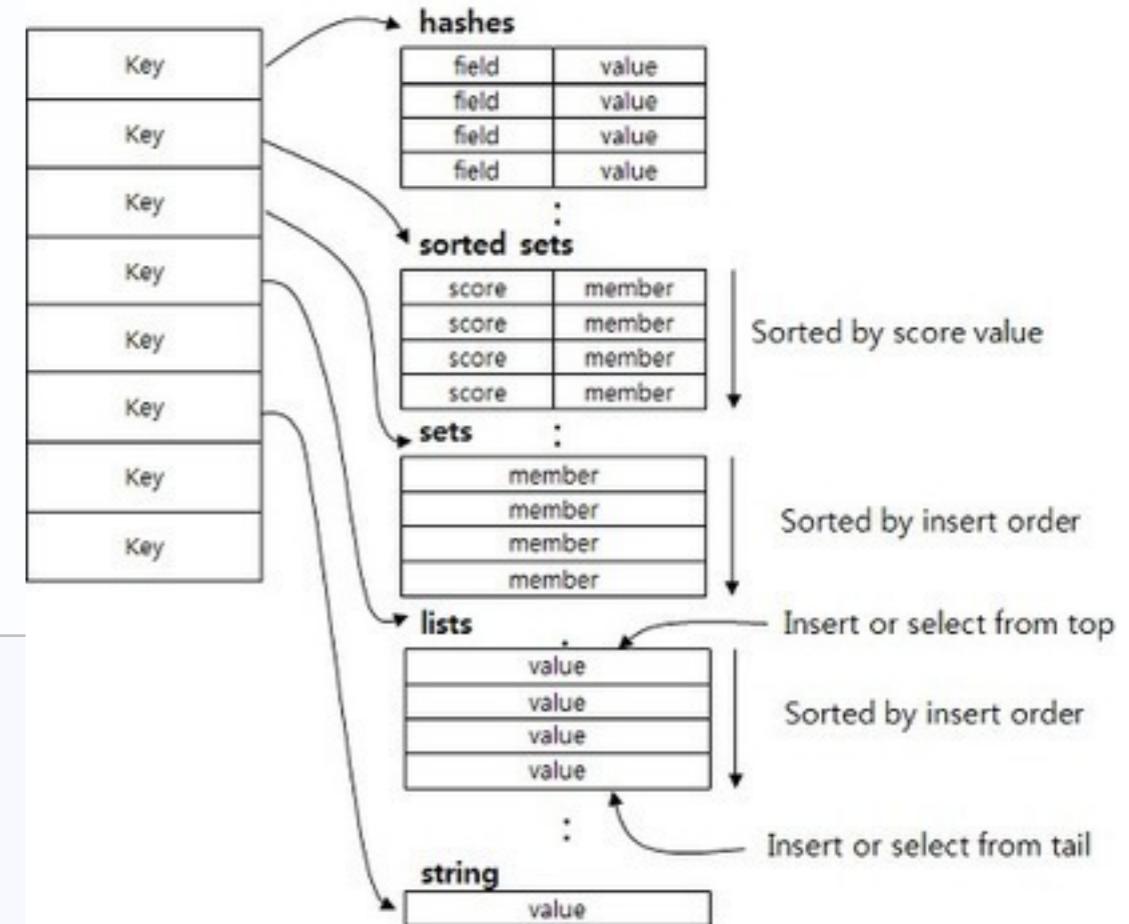
# NoSQL - Clé / valeur

## ► Exemple Redis

```
var redis = require("redis"),
    client = redis.createClient();

client.on("error", function (err) {
    console.log("Error " + err);
});

client.set("string key", "string val", redis.print);
client.hset("hash key", "hashtest 1", "some value", redis.print);
client.hset(["hash key", "hashtest 2", "some other value"], redis.print);
client.hkeys("hash key", function (err, replies) {
    console.log(replies.length + " replies:");
    replies.forEach(function (reply, i) {
        console.log("    " + i + ": " + reply);
    });
    client.quit();
});
```



# NoSQL - Orienté Colonne

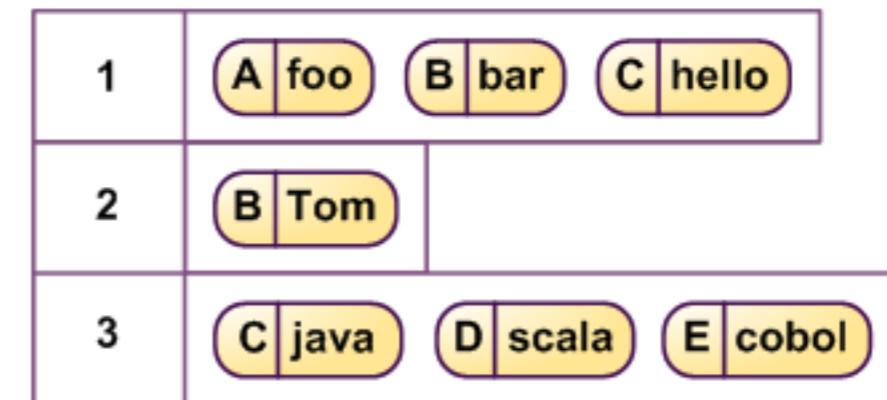


- Exemple Cassandra

```
var cassandra = require('cassandra-driver');
var client = new cassandra.Client({ contactPoints: ['h1', 'h2'], keyspace: 'ks1'});
var query = 'SELECT email, last_name FROM user_profiles WHERE key=?';
client.execute(query, ['guy'], function(err, result) {
  assert.ifError(err);
  console.log('got user profile with email ' + result.rows[0].email);
});
```

	A	B	C	D	E
1	foo	bar	hello		
2		Tom			
3			java	scala	cobol

Organisation d'une table dans une BDD relationnelle



Organisation d'une table dans une BDD orientée colonnes



# NoSQL - Orienté Document

## ► Exemple CouchDB

```
var cradle = require('cradle');
var db = new(cradle.Connection)().database('starwars');

db.get('vader', function (err, doc) {
  doc.name; // 'Darth Vader'
  assert.equal(doc.force, 'dark');
});

db.save('skywalker', {
  force: 'light',
  name: 'Luke Skywalker'
}, function (err, res) {
  if (err) {
    // Handle error
  } else {
    // Handle success
  }
});
```

```
db.users.insert ( ← collection
  {
    name: "sue", ← field: value
    age: 26, ← field: value
    status: "A" ← field: value
  }
)
```

The diagram illustrates the structure of a document insertion operation. It shows the code snippet above with annotations. A green arrow points from the word 'collection' to the 'db.users.insert' part of the code. Another green arrow points from the word 'field: value' to each of the three key-value pairs ('name: "sue"', 'age: 26', and 'status: "A"'). A large curly brace on the right side groups these three key-value pairs under the label 'document'.

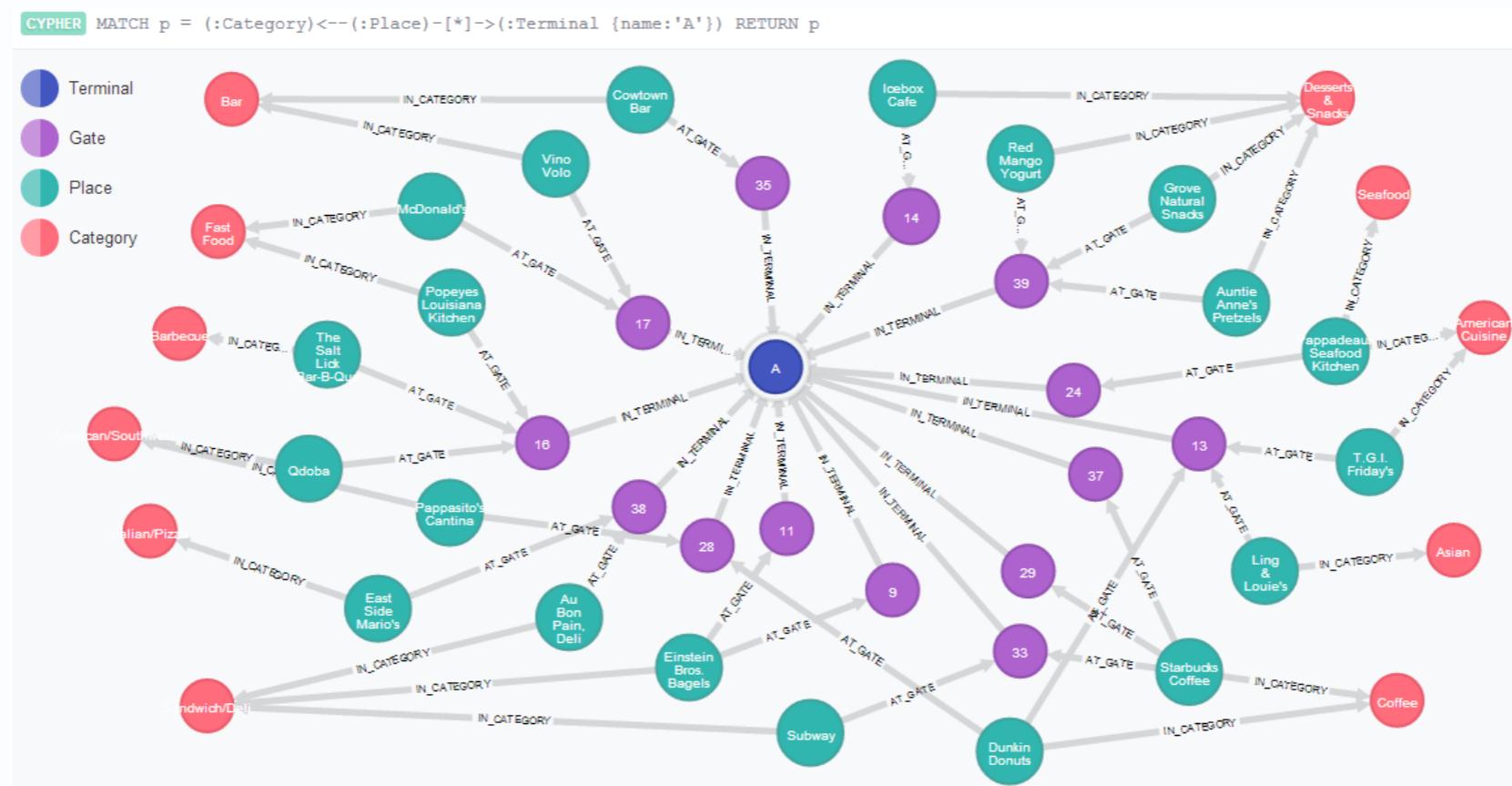


# NoSQL - Orienté Graphe

- Exemple neo4j

```
var r = require('request');

r.post({
  uri: 'http://localhost:7474/db/data/transaction/commit',
  json: {
    statements: [{
      statement: 'MATCH (n:User) RETURN n, labels(n) as l LIMIT {limit}',
      parameters: { limit: 10 }
    }]
  },
  function(err, res) { console.log(JSON.stringify(res.body)); }
);
```





- MongoDB
  - Base de données écrite en C++, inclus le moteur JS SpiderMonkey
- Document

MongoDB permet de manipuler des objets structurés au format BSON (JSON binaire). Les données prennent la forme de documents enregistrés eux-mêmes dans des collections.
- Accès aux données

L'accès aux données se fait via un protocole réseau, les requêtes sont décrites sous forme d'objet JavaScript
- Absence de Schéma

Contrairement à un SGBDR, les documents stockés dans une collection peuvent avoir des formats complètement différents. Les données peuvent également être imbriquées.

# NoSQL - MongoDB



- ▶ Installation

- Windows

- <https://www.mongodb.com/>

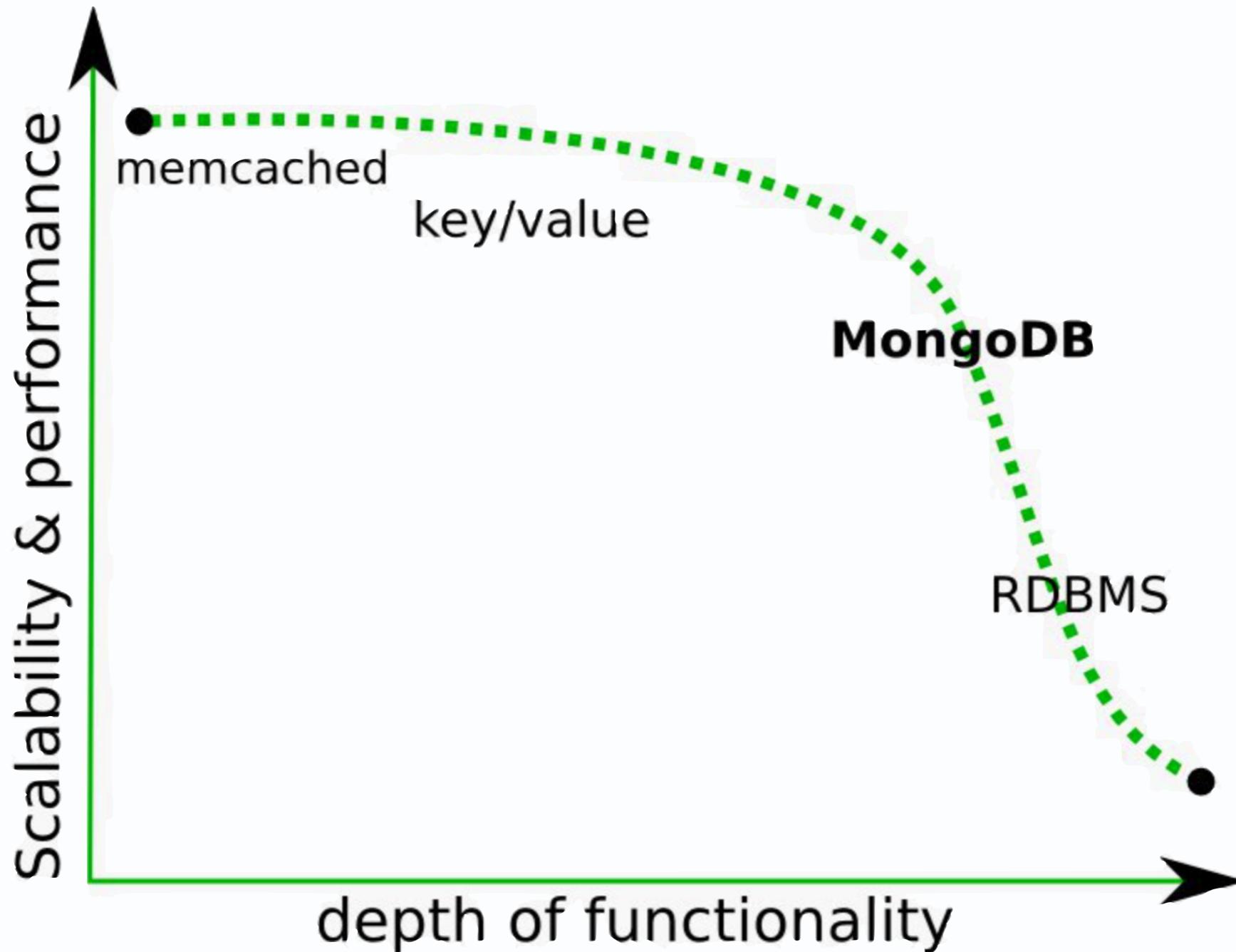
- <https://www.mongodb.org/dl/win32>

- Mac

- <https://www.mongodb.com/>

- brew install mongo

# NoSQL - MongoDB





# NoSQL - MongoDB

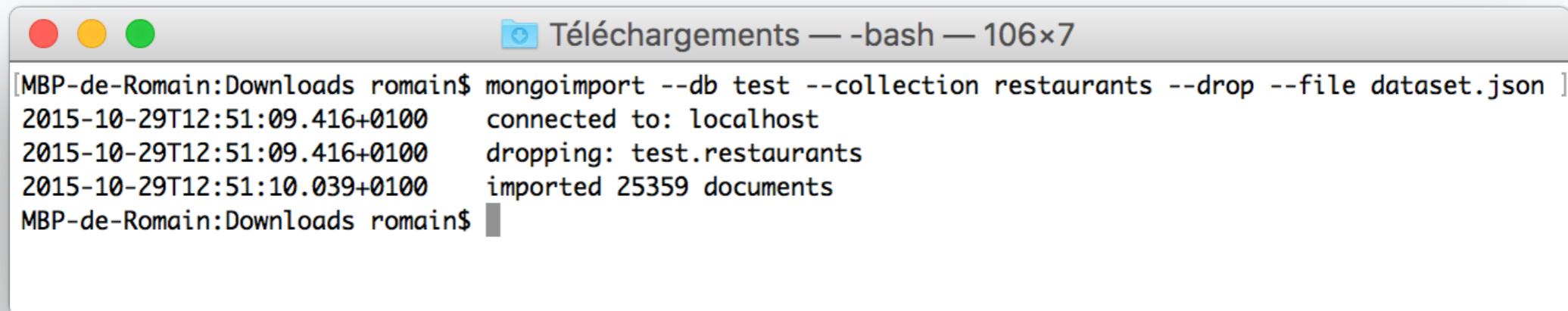
- ▶ Parallèle avec un SGBDR

MongoDB	SGBDR
Base de données	Base de données
Collection	Table
Document	Enregistrement
Pas de schéma	Schéma
API JavaScript	SQL



# NoSQL - MongoDB

- ▶ Jeu de données d'exemple fourni par Mongo  
<https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/primer-dataset.json>
- ▶ Importer un jeu de données  
mongoimport --collection restaurants --file ~/downloads/primer-dataset.json



```
[MBP-de-Romain:Downloads roman$ mongoimport --db test --collection restaurants --drop --file dataset.json ]  
2015-10-29T12:51:09.416+0100 connected to: localhost  
2015-10-29T12:51:09.416+0100 dropping: test.restaurants  
2015-10-29T12:51:10.039+0100 imported 25359 documents  
MBP-de-Romain:Downloads roman$
```

# NoSQL - MongoDB



## ▶ MongoShell

Mongo livre un programme client en ligne de commande pour accéder à la base.

```
[MBP-de-Romain:~ romain$ mongo
MongoDB shell version: 3.0.7
connecting to: test
[> use address_book
switched to db address_book
[> db.contact.find()
{ "_id" : ObjectId("562d4e878561c01ec2e43cfb"), "prenom" : "Steve", "nom" : "Jobs" }
{ "_id" : ObjectId("562d4e918561c01ec2e43cfb"), "prenom" : "Bill", "nom" : "Gates" }
{ "_id" : ObjectId("562d4eab8561c01ec2e43cfb"), "prenom" : "Mark", "nom" : "Zuckerberg" }
[> db.contact.insert({prenom: 'Steve', nom: 'Ballmer'})
WriteResult({ "nInserted" : 1 })
[> db.contact.find({prenom: 'Steve'})
{ "_id" : ObjectId("562d4e878561c01ec2e43cfb"), "prenom" : "Steve", "nom" : "Jobs" }
{ "_id" : ObjectId("562d4ee1321ac0f47f03ce9d"), "prenom" : "Steve", "nom" : "Ballmer" }
> ]]
```

# NoSQL - MongoDB



## ▶ Principales Commandes MongoShell

Shell Helpers	JavaScript Equivalents
show dbs, show databases	db.adminCommand('listDatabases')
use <db>	db = db.getSiblingDB('<db>')
show collections	db.getCollectionNames()
show users	db.getUsers()
show roles	db.getRoles({showBuiltInRoles: true})
show log <logname>	db.adminCommand({ 'getLog' : '<logname>' })
show logs	db.adminCommand({ 'getLog' : '*' })
it	cursor = db.collection.find() if ( cursor.hasNext() ){ cursor.next(); }

# NoSQL - MongoDB



- ▶ MongoClient
  - API officiel fourni MongoDB pour accéder aux données sous Node.js
- ▶ Installation
  - npm install mongodb --save
- ▶ Insertion

```
var MongoClient = require('mongodb').MongoClient;
var url = 'mongodb://localhost:27017/addressbook';

MongoClient.connect(url, function(err, db) {
  if (err) {
    console.log('Erreur : ' + err);
    return;
  }
  var cursor = db.collection('contacts').insert({prenom: 'Romain', nom: 'Bohdanowicz'},
function(err, result) {
  if (err) {
    console.log('Erreur : ' + err);
    return;
  }

  console.log('Le contact a bien été inséré');
});
});
```

# NoSQL - MongoDB



## ► Modification

```
var cursor = db.collection('contacts').update({nom: 'Bohdanowicz'}, {prenom: 'ROMAIN', nom: 'BOHDANOWICZ'}, {upsert:true}, function(err, result) {
  if (err) {
    console.log('Erreur : ' + err);
    return;
  }

  console.log('Le contact a bien été mis à jour');
});
```

## ► Suppression

```
var cursor = db.collection('contacts').removeOne({nom: 'BOHDANOWICZ'}, function(err, result) {
  if (err) {
    console.log('Erreur : ' + err);
    return;
  }

  console.log('Le contact a bien été supprimé');
});
```

# NoSQL - MongoDB



## ► Recherche

```
var MongoClient = require('mongodb').MongoClient;
var url = 'mongodb://localhost:27017/addressbook';

MongoClient.connect(url, function(err, db) {
  if (err) {
    console.log('Erreur : ' + err);
    return;
  }
  var cursor = db.collection('contacts').find();
  cursor.toArray(function(err, contacts) {
    console.log(contacts);
    db.close();
  });
});
```

# NoSQL - MongoDB



## ► Recherche multi-critères

Exemple : Restaurants de Brooklyn, ET dont la cuisine est française OU italienne ET dont l'une des notes est supérieur à 40

```
var MongoClient = require('mongodb').MongoClient;
var url = 'mongodb://localhost:27017/test';
MongoClient.connect(url, function(err, db) {
  if (err) {
    console.log('Erreur : ' + err);
    return;
  }
  var cursor = db.collection('restaurants').find({
    borough: 'Brooklyn',
    $or: [
      { "cuisine": "Italian" },
      { "cuisine": "French" },
      'grades.score': { $gt: 40 }
    ],
    cursor.toArray(function(err, restaurants) {
      restaurants.forEach(function(r) {
        console.log(`Nom : ${r.name}, cuisine : ${r.cuisine}, adresse : ${r.address.building} ${r.address.street}`);
      });
      db.close();
    });
  });
});
```

```
MBP-de-Romain:MongoClient romain$ node multicriteres.js
Nom : Doc Wine Bar, cuisine : Italian, adresse : 83 North 7 Street
Nom : Le Gamin, cuisine : French, adresse : 556 Vanderbilt Avenue
Nom : Peperoncino, cuisine : Italian, adresse : 72 5 Avenue
Nom : Patrizia'S, cuisine : Italian, adresse : 35 Broadway
Nom : Tutta Pasta, cuisine : Italian, adresse : 160 7 Avenue
Nom : Anella, cuisine : Italian, adresse : 222 Franklin Street
Nom : Joe'S Pizza, cuisine : Italian, adresse : 349 5 Avenue
MBP-de-Romain:MongoClient romain$
```

# NoSQL - MongoDB



- ▶ Mongoose  
ODM : Object Document Mapping, permet de communiquer avec Mongo avec des objets Entités
- ▶ Installation  
`npm install mongoose --save`
- ▶ Schema  
Mongo permet l'absence de schéma, ce qui est peu recommandable dans une utilisation sous la forme d'entité. Mongoose réintroduit ce concept.

# NoSQL - MongoDB



- ▶ Création d'un Schéma

```
var mongoose = require('mongoose');

var contactSchema = mongoose.Schema({
  firstName: String,
  lastName: String,
})

var Contact = mongoose.model('contact', contactSchema);
```

```
mongoose.connect('mongodb://localhost/addressbook');
var db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', function (callback) {
  var contacts = Contact.find(function (err, contacts) {
    if (err) return console.error(err);
    reply({data: contacts});
  });
});
```

# NoSQL - Exercice



- Créer un Model contact avec prenom, nom, email et téléphone
- Utiliser Mongoose pour remplacer le tableau dans l'API REST créé précédemment
- Ajouter des validateurs sur le prenom et nom (champs obligatoire)
- Optionnel : Créer un model Société et lier les Model Société et Contact



**formation.tech**

# Tests Automatisés



# Tests automatisés - Introduction

## ▶ Vérification manuelle

- Ecrire une recette de tests et demander à une personne de la rejouer à des étapes clés (nouvelle version)
- Ecrire le test sous la forme de code, et vérifier visuellement que les résultats attendus soit les bons

## ▶ Tests automatisés

- Le test est codé, la vérification se fait dans un rapport

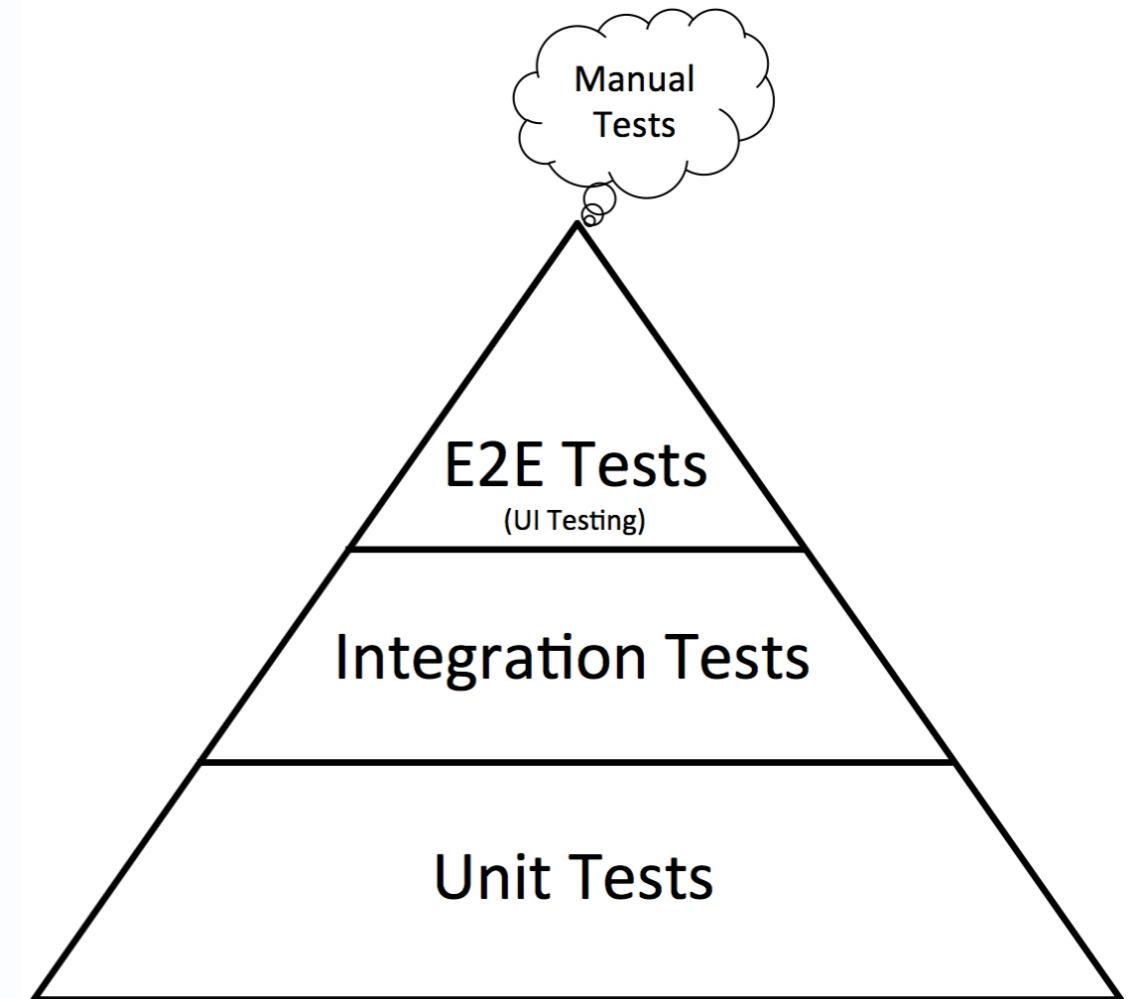
## ▶ Historique

- sUnit en 1994 (SmallTalk), JUnit en 1997 (Java)
- Les frameworks s'inspirant de jUnit sont catégorisés xUnit (PHPUnit, CUnit...)



# Tests automatisés - Pyramide des Tests

- ▶ Types de tests
  - **Unitaire** : tests des méthodes d'une classe
  - **Intégration** : teste l'intégration entre plusieurs classes
  - **Fonctionnels** : teste l'application du point de vue du client (HTTP dans le cas du web)
  - **End-to-End (E2E)** : teste l'application dans le client (y compris JavaScript, CSS...)





# Tests automatisés - Karma



- ▶ Lanceur de test

Permet de lancer vos tests simultanément dans Chrome, Firefox, Internet Explorer...

- ▶ Installation

npm install -g karma-cli

npm install karma —save-dev

- ▶ Configuration du projet

karma init

- ▶ Lancement des tests

karma start

```
Air-de-Romain:Jasmine romain$ karma init

Which testing framework do you want to use ?
Press tab to list possible options. Enter to move to the next question.
> jasmine

Do you want to use Require.js ?
This will add Require.js plugin.
Press tab to list possible options. Enter to move to the next question.
> no

Do you want to capture any browsers automatically ?
Press tab to list possible options. Enter empty string to move to the next question.
> Chrome
> Safari
>

What is the location of your source and test files ?
You can use glob patterns, eg. "js/*.js" or "test/**/*Spec.js".
Enter empty string to move to the next question.
> |
```

```
Air-de-Romain:Jasmine romain$ karma start
02 09 2015 21:30:11.510:INFO [karma]: Karma v0.13.9 server started at http://localhost:9876/
02 09 2015 21:30:11.518:INFO [launcher]: Starting browser Chrome
02 09 2015 21:30:11.526:INFO [launcher]: Starting browser Safari
02 09 2015 21:30:12.723:INFO [Safari 8.0.7 (Mac OS X 10.10.4)]: Connected on socket HE38s1HTBKXL5t5yAAAA with id 54715269
Safari 8.0.7 (Mac OS X 10.10.4): Executed 1 of 1 SUCCESS (0.038 secs / 0.003 secs)
Safari 8.0.7 (Mac OS X 10.10.4): Executed 1 of 1 SUCCESS (0.038 secs / 0.003 secs)
Chrome 45.0.2454 (Mac OS X 10.10.4): Executed 1 of 1 SUCCESS (0.04 secs / 0.008 secs)

TOTAL: 2 SUCCESS
```



# Tests automatisés - QUnit

- Crée en 2008 par les développeurs de jQuery
- Type xUnit (JUnit, PHPUnit...) : basés sur des assertions
- Plutôt destiné à du code client
- Installation
  - npm install --save-dev qunitjs
  - bower install --save-dev qunit
- Lancement des tests
  - Ouverture du fichier .html
  - grunt-contrib-qunit
  - karma-qunit





# Tests automatisés - QUnit

```
<!-- runner.html -->
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>QUnit Example</title>
  <link rel="stylesheet" href="node_modules/qunitjs/qunit/qunit.css">
</head>
<body>
  <div id="qunit"></div>
  <div id="qunit-fixture"></div>
  <script src="calculette.js"></script>
  <script src="node_modules/qunitjs/qunit/qunit.js"></script>
  <script src="calculette-test.js"></script>
</body>
</html>
```

```
// calculette-test.js
QUnit.test("Test addition", function(assert) {
  assert.equal(calculette.ajouter(2, 3), 5, "2 + 3 = 5");
});
```

The screenshot shows a web browser window titled "QUnit Example". The address bar indicates the URL is "localhost:63342/JavaScript/Tests/QUnit/testCalculette.html". The main content area displays the QUnit test results:

- QUnit Example**
- Test results:
  - Hide passed tests
  - Check for Globals
  - No try-catch
  - Filter:  Go
- Test summary:
  - QUnit 1.19.0; Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_10\_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.85 Safari/537.36
  - Tests completed in 7 milliseconds.
  - 1 assertions of 1 passed, 0 failed.
- Test details:
  - 1. Test addition (1) Rerun
    - 1. 2 + 3 = 5 @ 1 ms
- Source: at <http://localhost:63342/JavaScript/Tests/QUnit/testCalculette.html:14:11>



# Tests automatisés - Jasmine

- Crée en 2010
- Type BDD (Behavior-Driven Development)
- Fonctionne pour le browser ou node.js
- Installation et lancement des tests (node)  
`npm install -g jasmine`  
`jasmine init`  
`jasmine`
- Installation et lancement des tests (browser)  
`npm install --save-dev jasmine-core`  
`SpecRunner.html`  
`karma`





# Tests automatisés - Jasmine

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Jasmine Spec Runner v2.3.4</title>

  <link rel="shortcut icon" type="image/png" href="node_modules/jasmine-core/images/jasmine_favicon.png">
  <link rel="stylesheet" href="node_modules/jasmine-core/lib/jasmine-core/jasmine.css">

  <script src="node_modules/jasmine-core/lib/jasmine-core/jasmine.js"></script>
  <script src="node_modules/jasmine-core/lib/jasmine-core/jasmine-html.js"></script>
  <script src="node_modules/jasmine-core/lib/jasmine-core/boot.js"></script>

  <!-- include source files here... -->
  <script src="calculette.js"></script>

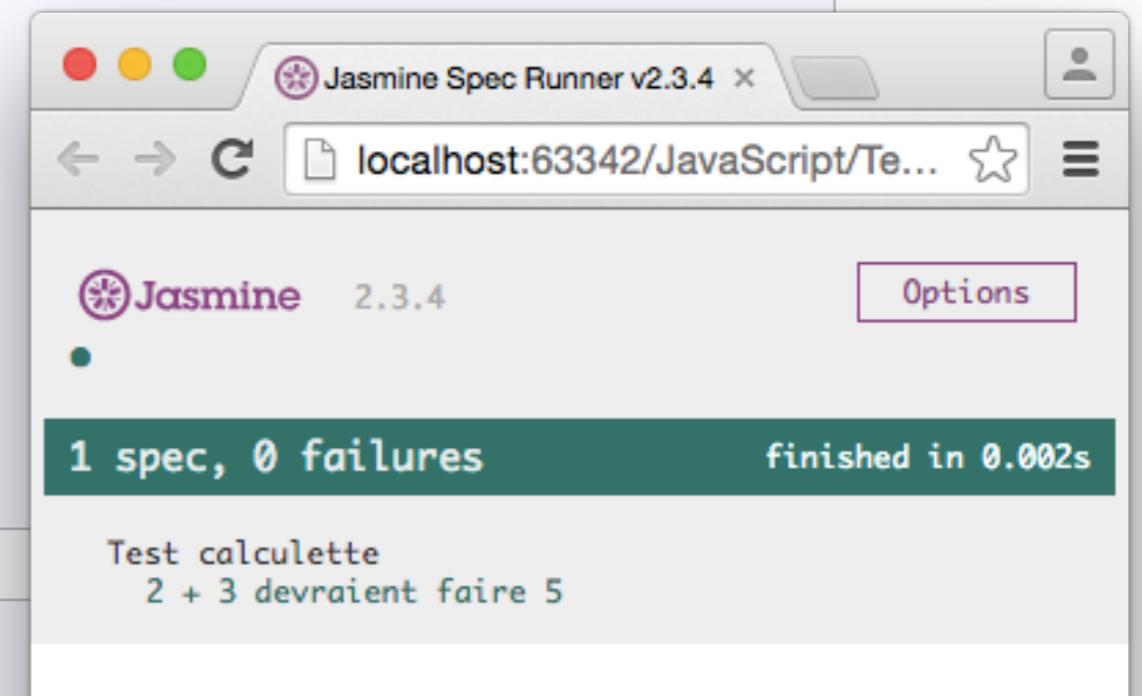
  <!-- include spec files here... -->
  <script src="spec/CalculetteSpec.js"></script>
</head>

<body>
</body>
</html>
```

```
describe("Test calculette", function() {

  it("2 + 3 devraient faire 5", function() {
    expect(calculette.ajouter(2, 3)).toEqual(5);
  });

});
```



# Tests automatisés - Mocha



- Crée en 2011
- Type assert ou BDD (le framework est flexible)
- Fonctionne pour le browser ou node.js
- Installation et lancement des tests (node)  
`npm install -g mocha`  
`mocha`
- Installation et lancement des tests (browser)  
`npm install -g mocha`  
`mocha init`  
`npm install chai`  
`runner.html`  
`karma`





# Tests automatisés - Mocha

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mocha</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="mocha.css" />
  </head>
  <body>
    <div id="mocha"></div>
    <script src="mocha.js"></script>
    <script src="node_modules/chai/chai.js"></script>
    <script>mocha.setup('bdd');</script>
    <script src="src/calcullette.js"></script>
    <script src="test/calcullette-test.js"></script>
    <script>
      mocha.run();
    </script>
  </body>
</html>
```

```
var assert = chai.assert;

describe('Test Addition', function() {
  it('2 + 3 devraient faire 5', function () {
    assert.equal(5, calcullette.ajouter(2, 3));
  });
});
```





# Tests automatisés - Voir aussi

- ▶ Coverage :
  - Istanbul : <https://istanbul.js.org>
- ▶ Doubles :
  - Sinon : <http://sinonjs.org>
- ▶ Parallélisation des tests :
  - Jest : <https://facebook.github.io/jest/>
  - AVA : <https://github.com/avajs/ava>
- ▶ Tests End-to-End :
  - Selenium : <http://www.seleniumhq.org>
  - Webdriver : <http://webdriver.io>
- ▶ PAAS de tests :
  - Sauce Labs : <https://saucelabs.com>
  - Browser Stack : <https://www.browserstack.com>