Formation Node.js, programmation JavaScript côté serveur

Romain Bohdanowicz
Twitter : @bioub - Github : https://github.com/bioub
http://formation.tech/

- Introduction
- Rappels JavaScript
- Node.js
- Modules JavaScript
- Gestion de dépendances
- NoSQL
- Express
- Grunt
- Framework HTML/CSS/JS
- Angular
- MEAN

Introduction

Présentations

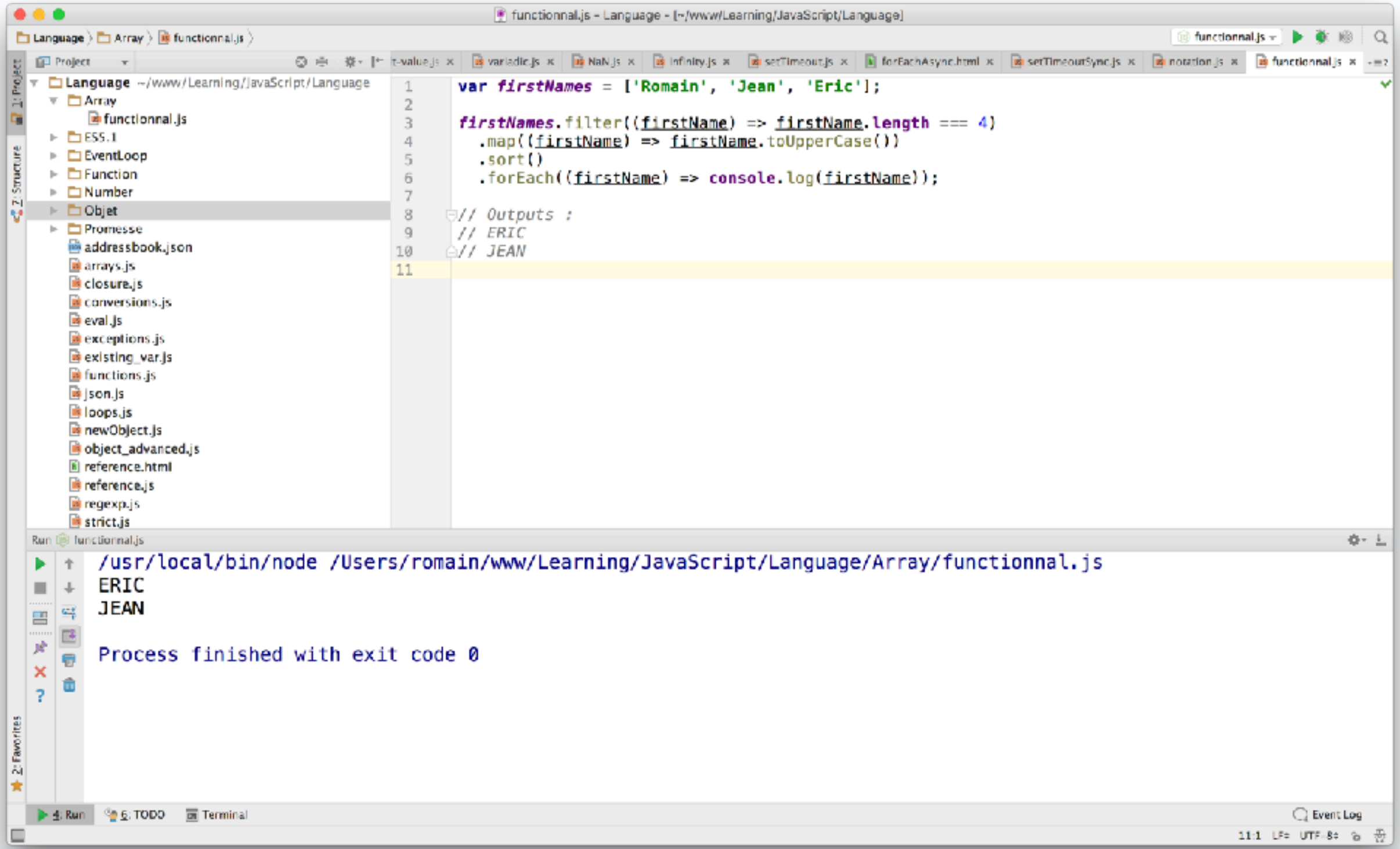
- ## Romain Bohdanowicz
  Ingénieur EFREI 2008, spécialité en Ingénierie Logicielle
- ## Expérience
  Formateur/Développeur Freelance depuis 2006
  Plus de 8000 heures de formation animées
- ## Langages
  Expert : HTML / CSS / JavaScript / PHP / Java
  Notions : C / C++ / Objective-C / C# / Python / Bash / Batch
- ## Certifications
  PHP 5 / PHP 5.3 / PHP 5.5 / Zend Framework 1
- ## Particularités
  Premier site web à 12 ans (HTML/JS/PHP), Triathlète à mes heures perdues
- ## Et vous ?
  Langages ? Expérience ? Utilité de cette formation ?

JavaScript IDEs

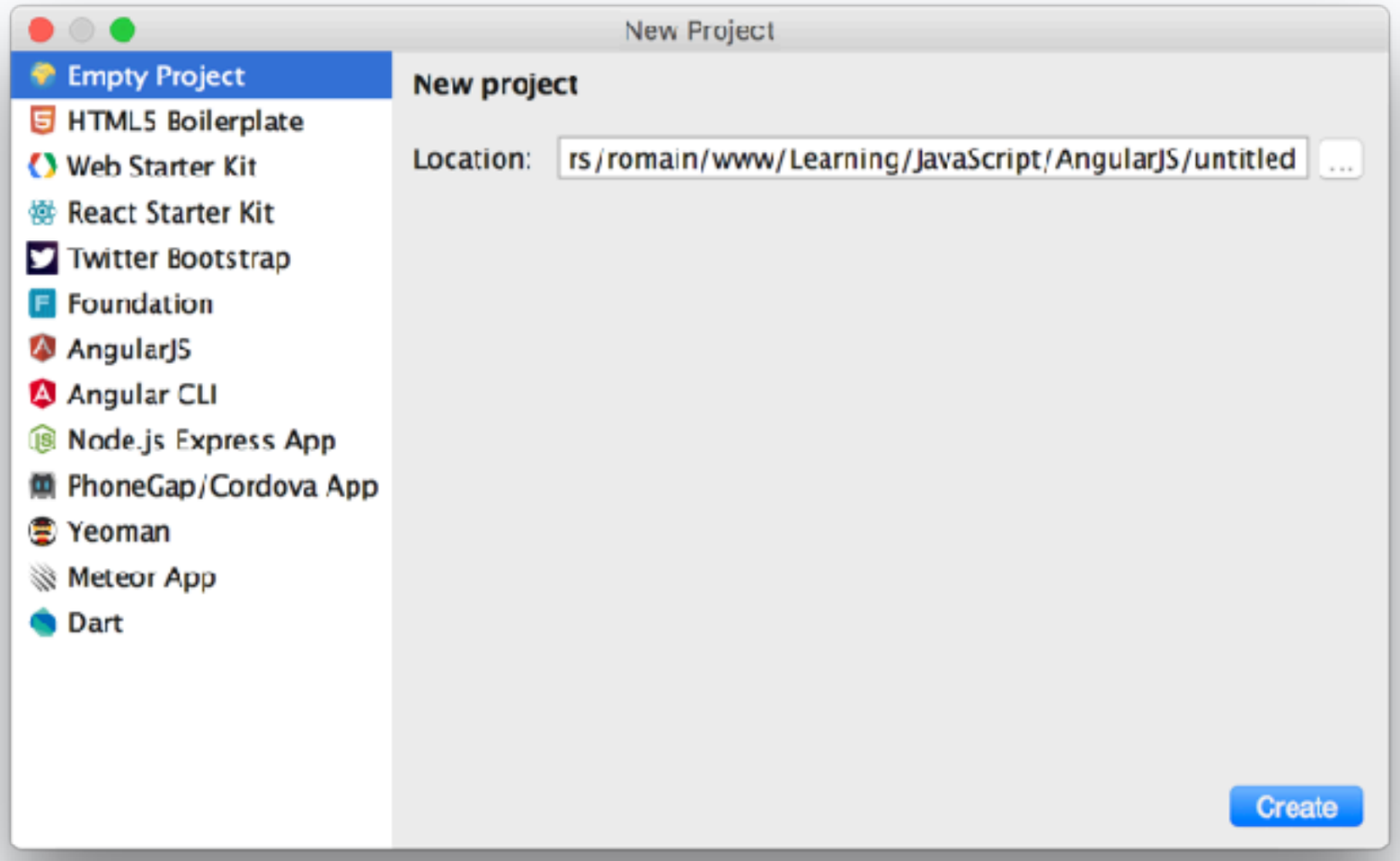
JavaScript IDEs - Webstorm

- 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

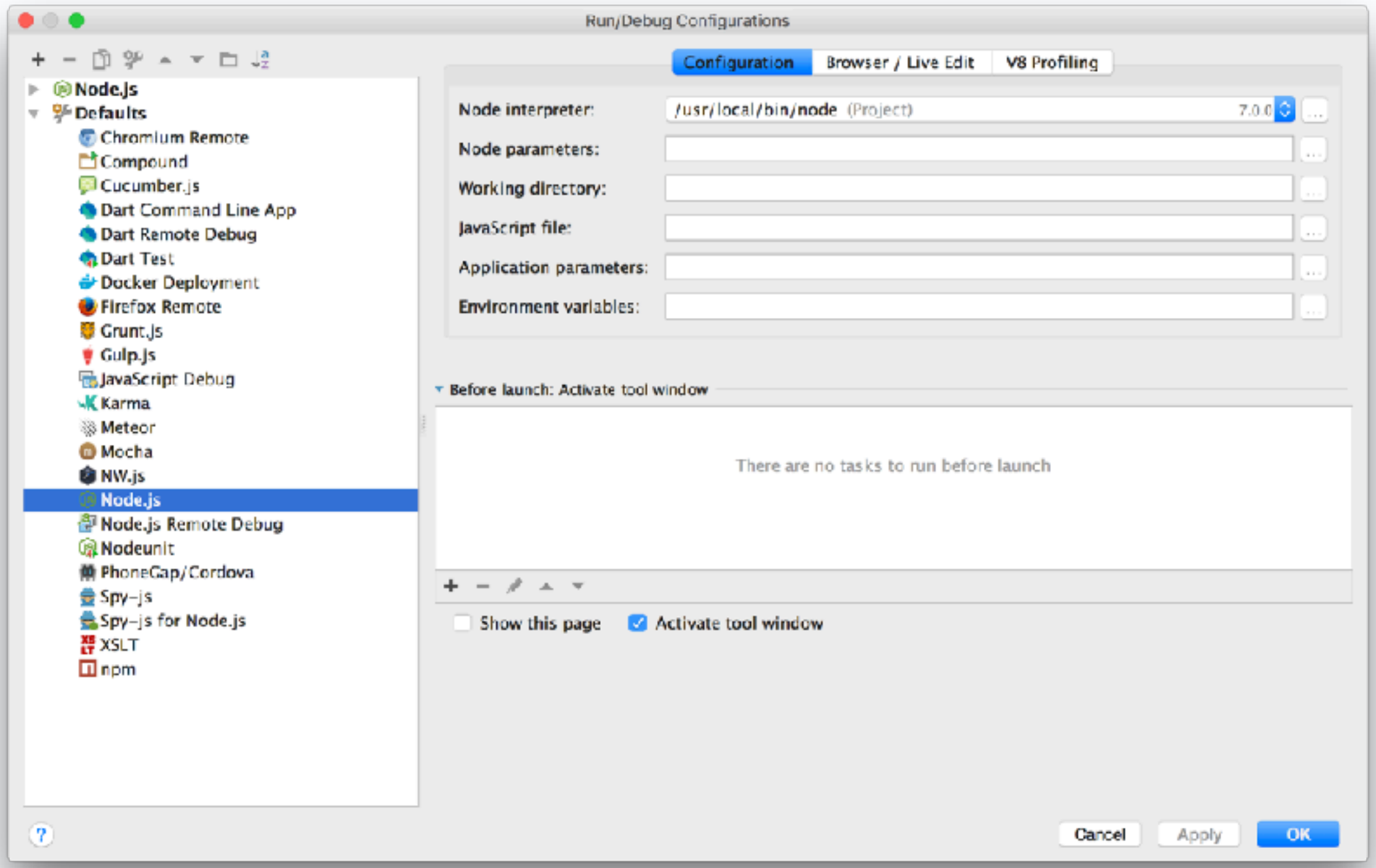
JavaScript IDEs - Webstorm



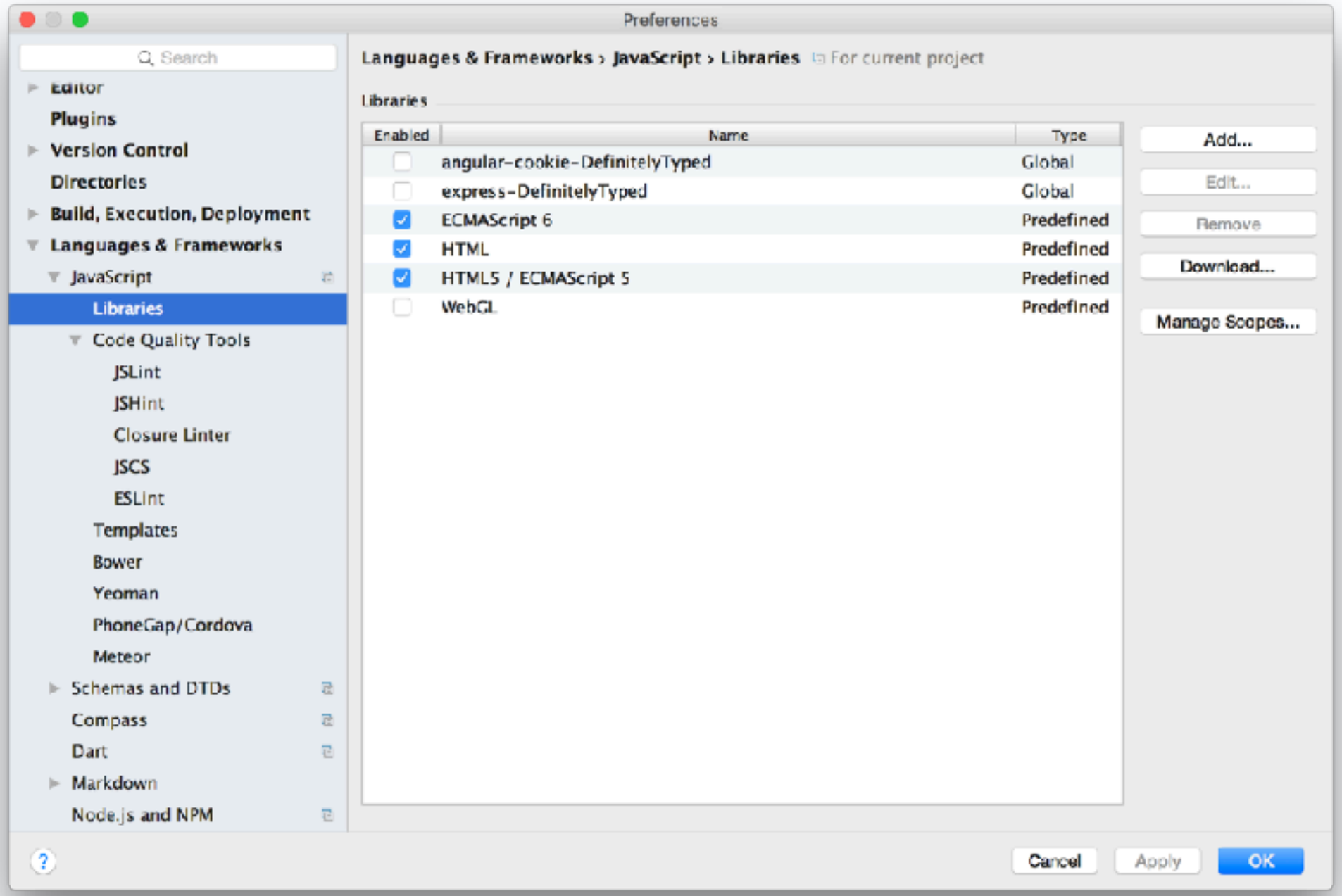
JavaScript IDEs - Webstorm



JavaScript IDEs - Webstorm



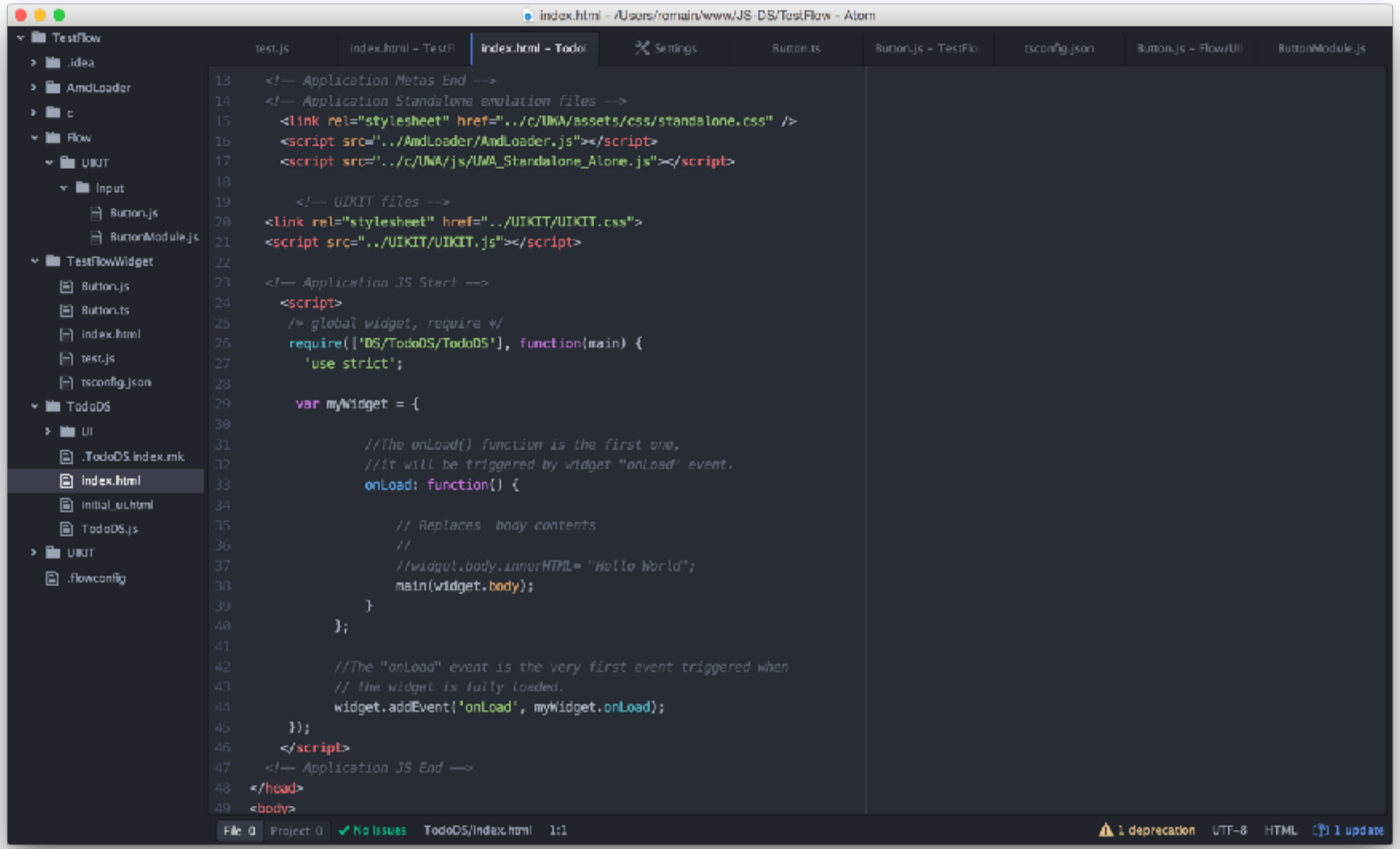
JavaScript IDEs - Webstorm



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…)

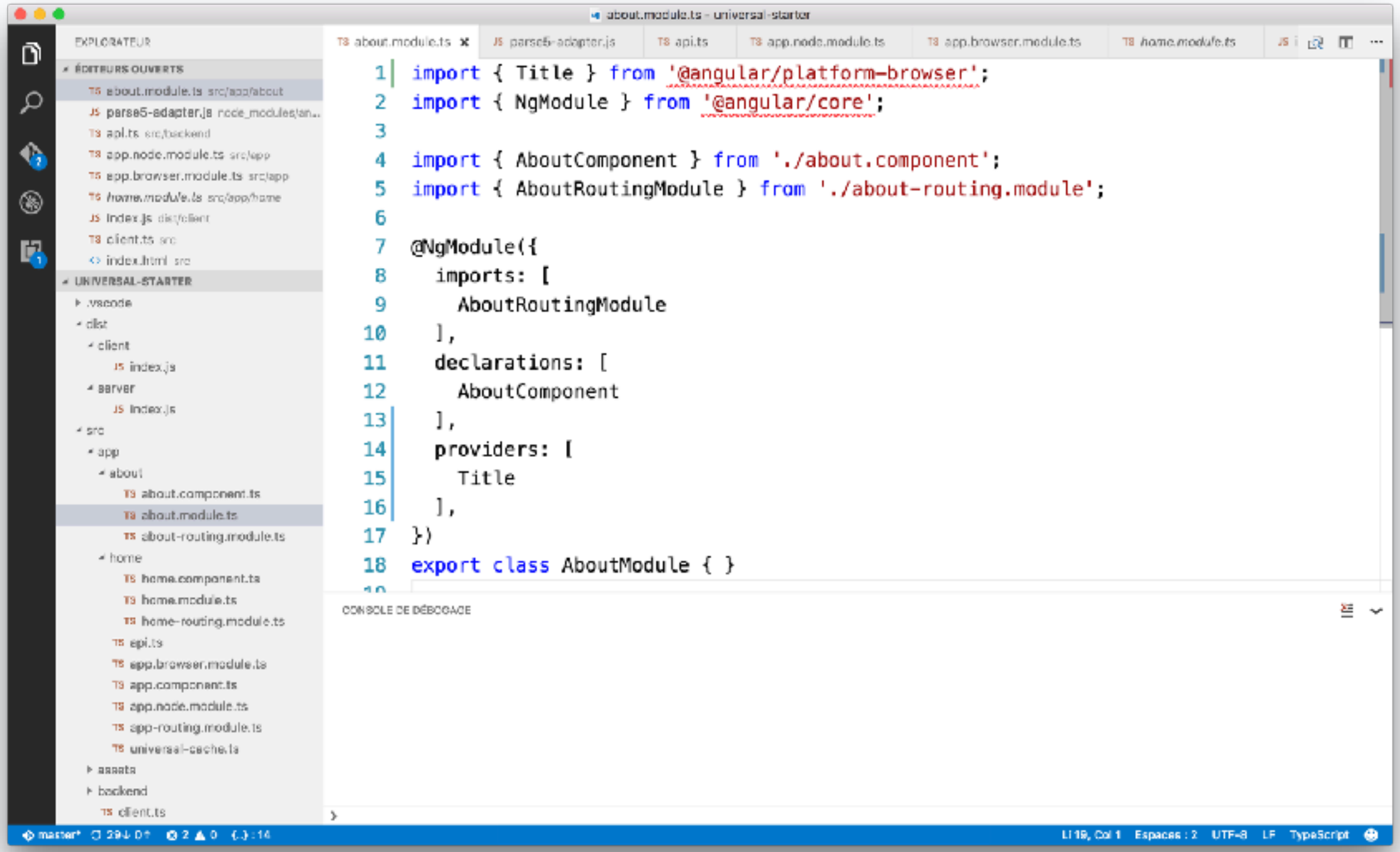
JavaScript IDEs - Atom



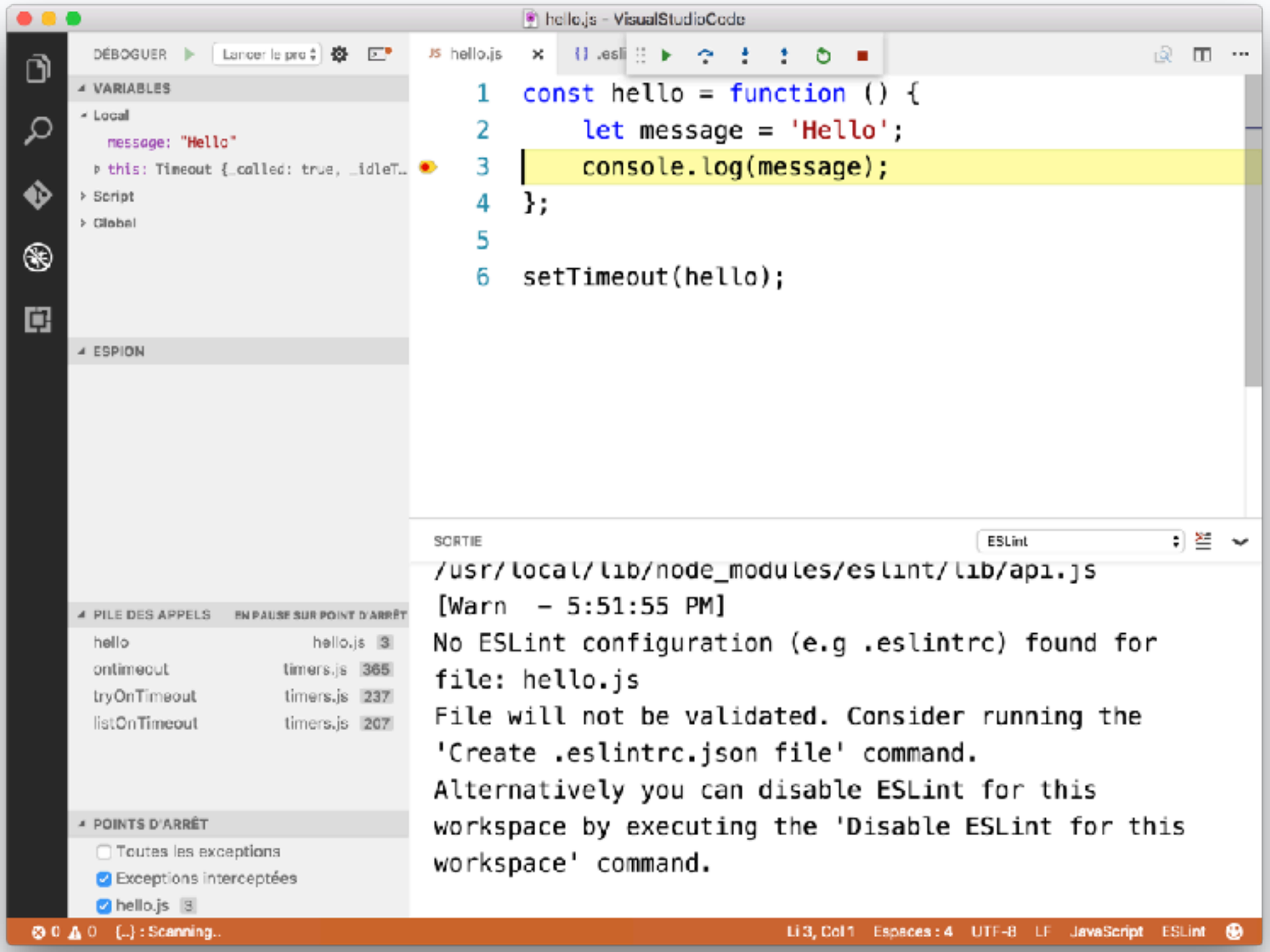
JavaScript IDEs - Visual Studio Code

- 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/VSCode
  Langage de création : JavaScript sous Node.js
- Documentation
  https://code.visualstudio.com/docs

JavaScript IDEs - Visual Studio Code



JavaScript IDEs - Visual Studio Code



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
```

JavaScript

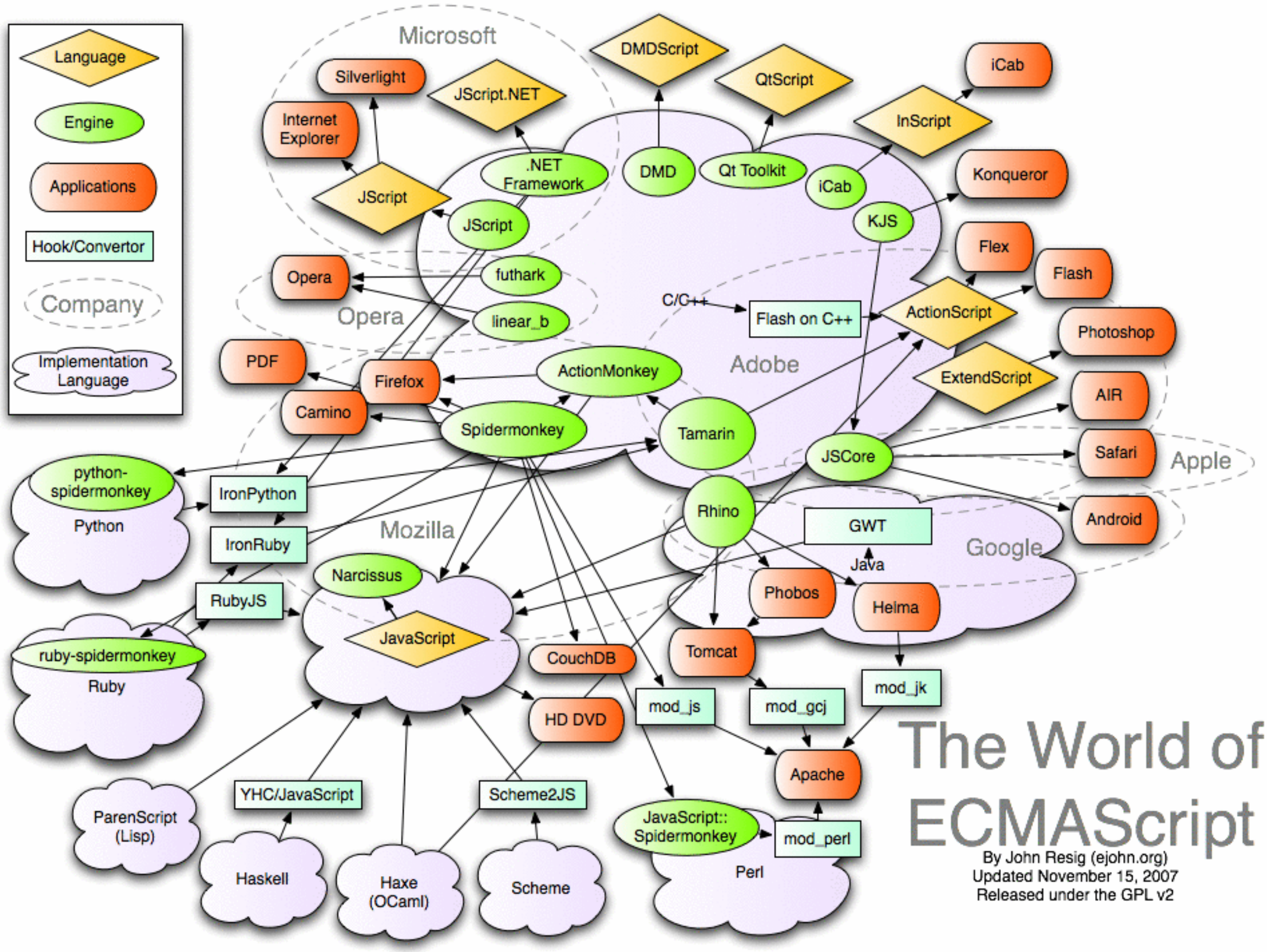
JavaScript - Introduction

- 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 créé en 1997 : ECMAScript

JavaScript - ECMAScript

- JavaScript est une implémentation de la norme ECMAScript 262
- La norme la plus récente est ECMAScript 2017
  Aussi appelée ECMAScript 8 ou ES8 (juin 2017)
  https://www.ecma-international.org/ecma-262/8.0/
- 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 - ECMAScript



JavaScript - Documentation

- La norme manque d'exemples et d'information sur les implémentations :
  http://www.ecma-international.org/ecma-262/7.0/
- 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 - 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 termine au choix par un point-virgule ou un retour à la ligne (même si les conventions incitent à la l’utilisation du point-virgule)
- 3 types de commentaires
  - // le commentaire s’arrête à la fin de la ligne
  - /* commentaire ouvrant/fermant */
  - /** Documentation JSDoc */

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 par des $ ou _ sont utiliser par certaines conventions
- Exemples :
  - Valides
    i, maVariable, $div, v1, prénom
  - Invalides
    1var, ma-variable

JavaScript - Mots clés

- Mots clés (ES8) :
  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 = 'Romain';
  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 (convertions, 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

```
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 - Opérateurs

- Affectation

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

JavaScript - Opérateurs

- Comparaison

Opérateur	Description	Exemples qui renvoient true

Égalité (==)	Renvoie true si les opérandes sont égaux après conversion en valeurs de mêmes types.	`3 == var1`
		`"3" == var1`
		`3 == '3'`

Inégalité (!=)	Renvoie true si les opérandes sont différents.	`var1 != 4` `var2 != "3"`
Égalité stricte (===)	Renvoie true si les opérandes sont égaux et de même type. Voir Object.is() et égalité de type en JavaScript.	`3 === var1`
Inégalité stricte (!==)	Renvoie true si les opérandes ne sont pas égaux ou s'ils ne sont pas de même type.	`var1 !== "3"` `3 !== '3'`
Supériorité stricte (>)	Renvoie true si l'opérande gauche est supérieur (strictement) à l'opérande droit.	`var2 > var1` `"12" > 2`
Supériorité ou égalité (>=)	Renvoie true si l'opérande gauche est supérieur ou égal à l'opérande droit.	`var2 >= var1` `var1 >= 3`
Infériorité stricte (<)	Renvoie true si l'opérande gauche est inférieur (strictement) à l'opérande droit.	`var1 < var2` `"2" < "12"`
Infériorité ou égalité (<=)	Renvoie true si l'opérande gauche est inférieur ou égal à l'opérande droit.	`var1 <= var2` `var2 <= 5`

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 renvoie2, 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)	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/incrémentation	`! ~ - + ++ -- 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'
```

JavaScript - API

- Standard Built-in Objects
  Les objets prédéfinies par le langages, voir la doc de Mozilla pour une liste exhaustive
  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

Fonctions en JavaScript - Introduction

- JavaScript est très consommateur de fonctions
  - réutilisation / factorisation
  - récursivité
  - fonction de rappel / écouteur
  - 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

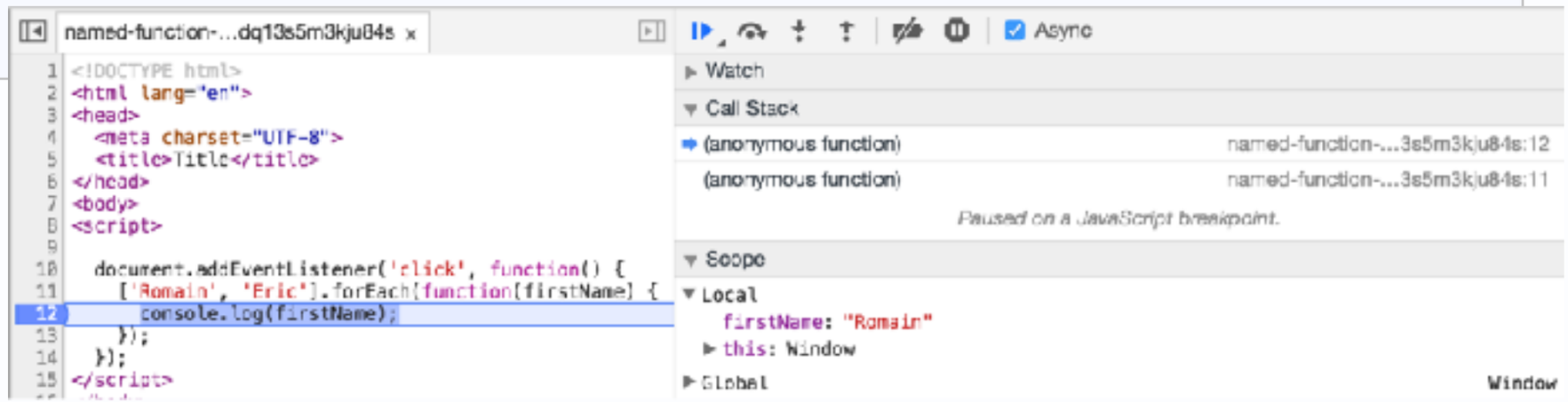
- En ES6 on pourrait même empêcher la redéclaration grace au mot clé const

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

console.log(hello());

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

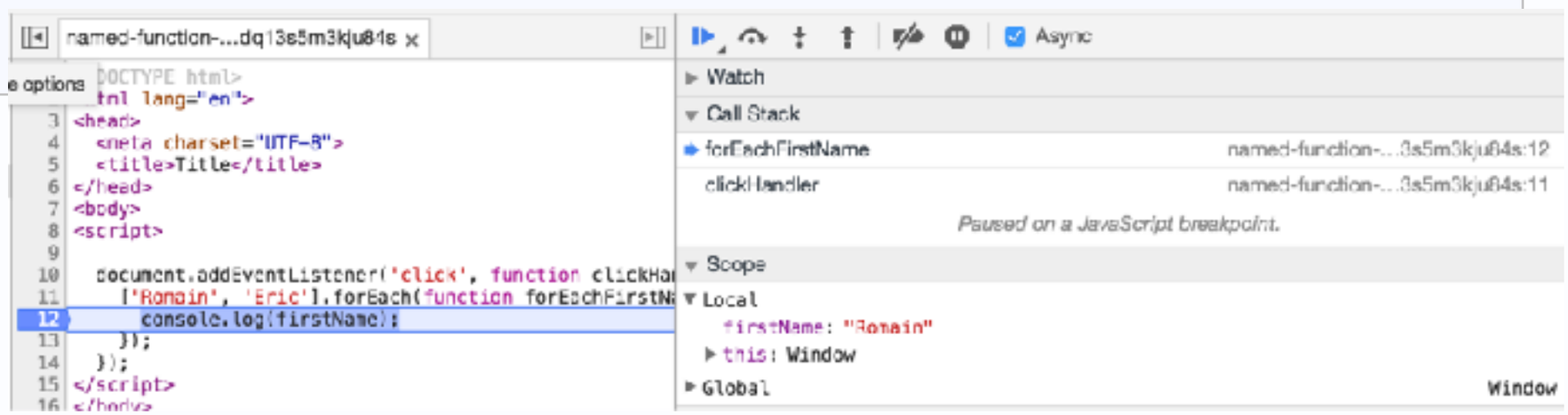
Fonctions en JavaScript - Named Function Expression

- Anonymous function expression vs Named function expression

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




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



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 lancer 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 (e) {
  console.log(e.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
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.

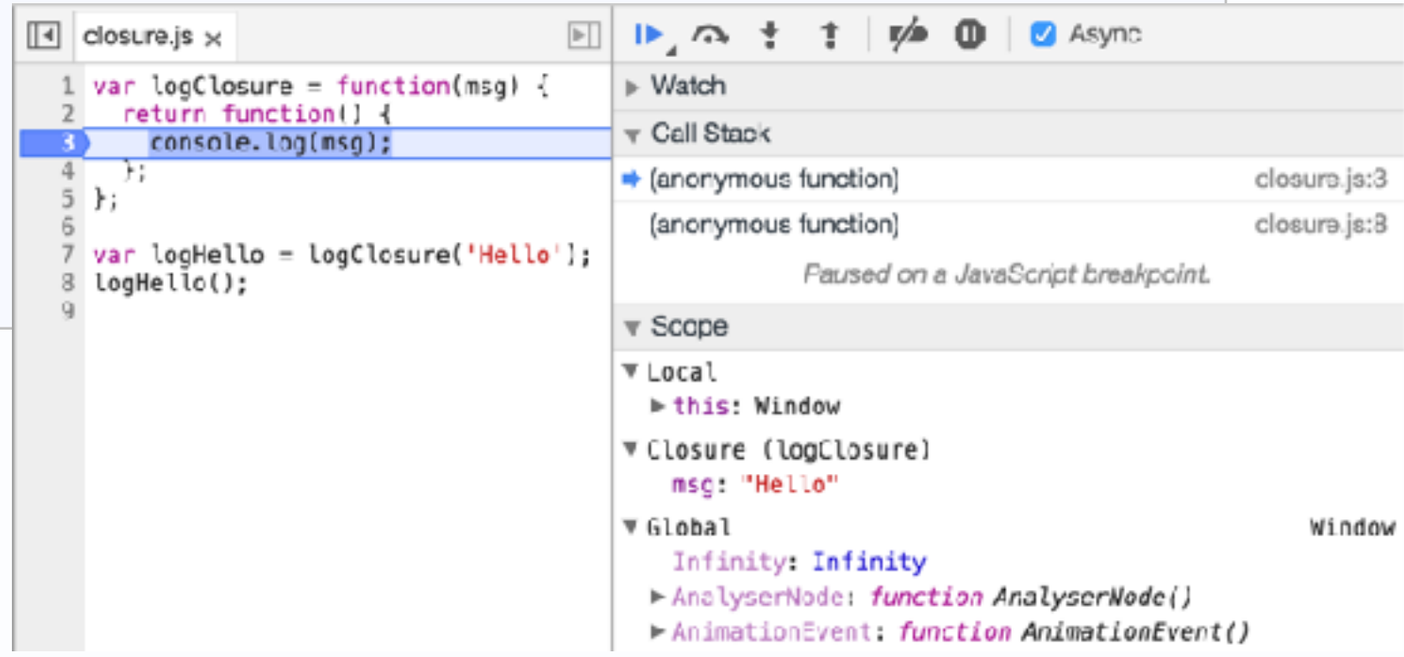
```
var a = function() {
  var b = function() {
    var c = function() {
      console.log(typeof a); // function
      console.log(typeof b); // function
      console.log(typeof c); // function
    };
    c();
  };
  b();
};
a();
```

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.

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

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



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

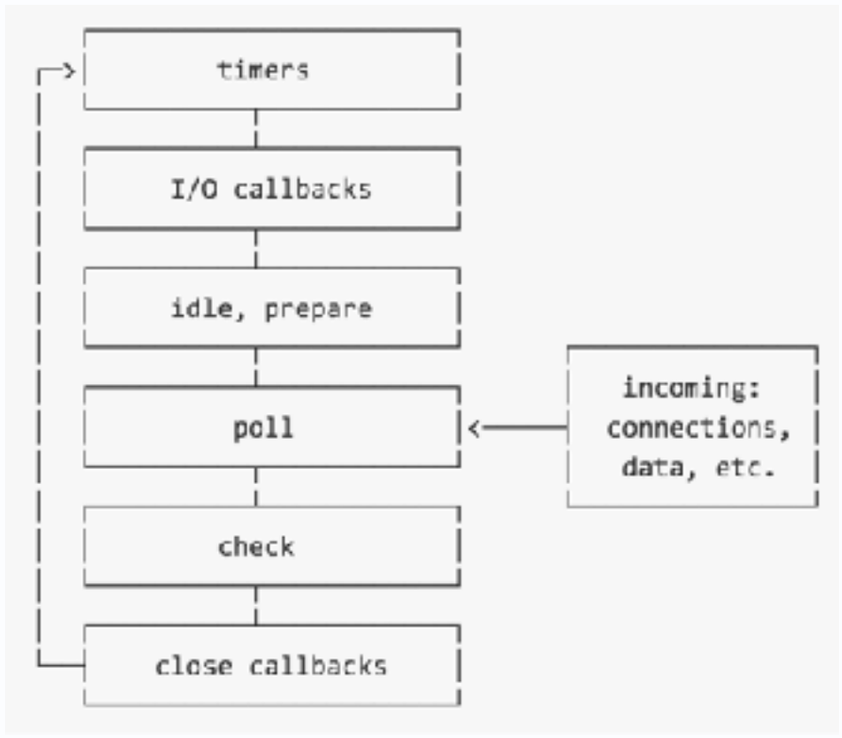
- Les moteurs JS sont par défaut mono-thread et mono-processus, ils ne peuvent donc exécuter qu’une seule tâche à la fois.
- Une boucle d’événements permet de passer d’un callback à l’autre de manière très performante, ex : traiter le clic d’un bouton entre 2 étapes d’une animation
- JavaScript est non-bloquant, 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

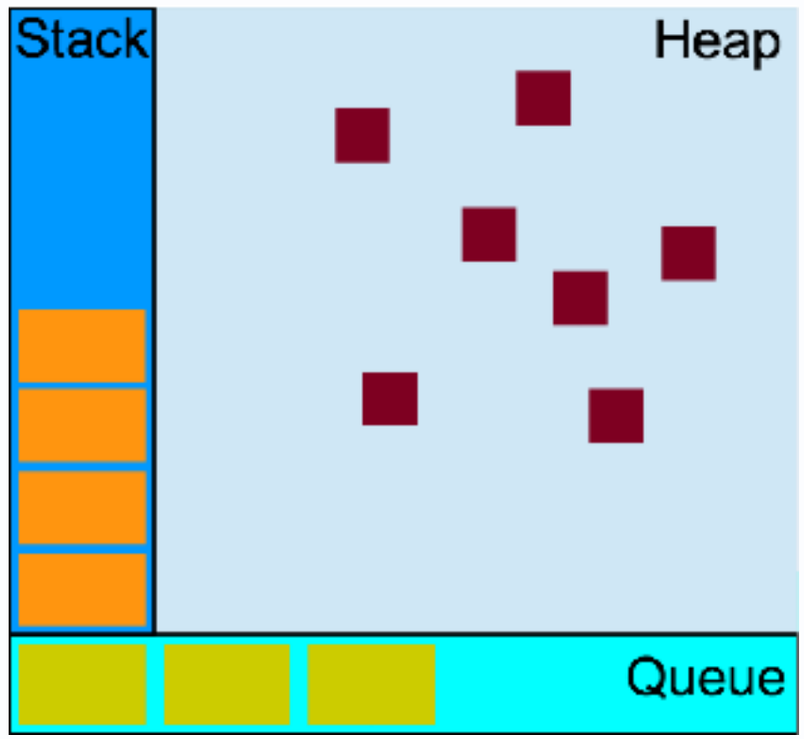
```
setTimeout(function() {
    console.log('1 fois dans 3 secondes');
}, 3000);

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.

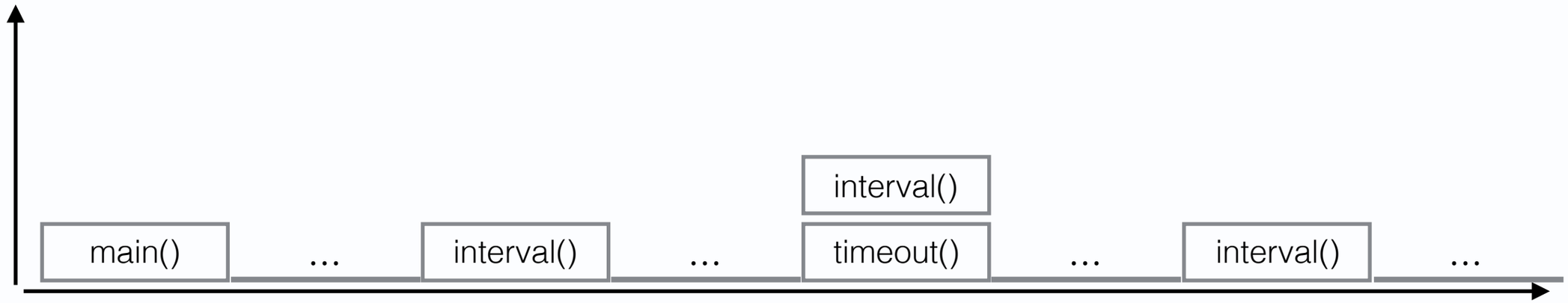
- What the heck is the event loop anyway? | JSConf EU 2014
  https://www.youtube.com/watch?v=8aGhZQkoFbQ





Fonctions en JavaScript - Boucle d’événements

- Boucle d’événements




```
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)
  Lorsque

```
(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 tentative (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.

JavaScript Orienté Objet

JavaScript Orienté Objet - Introduction

- Paradigme

  Par opposition à un modèle objet orienté classe, le modèle objet de JavaScript est orienté prototype

- Classe

  La notion de classe ou d'interface n'existe pas (seulement dans les docs où sous la forme de sucre syntaxique)

- Modèle statique vs Modèle dynamique

  Il n'y a pas de définition statique du type d'un objet, l'ajout de propriété où de méthode se fait dynamiquement à la création de l'objet

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 :

```
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) :

```
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 identifieurs 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 et 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

- 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
```

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 = ...;

const Jeu = ...;

const jeu = new Jeu({
  min: 0,
  max: 100
});

jeu.jouer();
```

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

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 - 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
Object.preventExtensions	Oui	Non	Non
Object.seal	Oui	Oui	Non
Object.freeze	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);
```

ECMAScript 6

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 ou ES7 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

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

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

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

ECMAScript 6 - Portées de bloc fonction

- 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 - new.target

- A l'instar de arguments, new.target est créé automatiquement lors de l'appel à une fonction fait avec new
- Contient la fonction utilisé ou undefined si pas d'appel avec new

```
var Contact = function() {
  console.log(new.target);
};

var c1 = new Contact(); // [Function: Contact]
var c2 = Contact(); // undefined
```

ECMAScript 6 - Fonctions fléchées

- Les fonctions fléchés sont plus courtes syntaxiquement

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

firstNames.filter(firstName => firstName.length === 4)
  .map(firstName => firstName.toUpperCase())
  .sort()
  .reverse()
  .reduce((firstNames, firstName) => {
    if (!firstNames.includes(firstName)) {
      firstNames.push(firstName)
    }
    return firstNames;
  }, [])
  .forEach(firstName => console.log(firstName));
```

ECMAScript 6 - Fonctions fléchées

- Les fonctions fléchés ne lient pas les variables this, arguments ou new.target
- Elles ne doivent pas être utilisée pour déclarer des méthodes !

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

contact.helloAsyncFunctionExpression(); // Hello my name is undefined
contact.helloAsyncArrow(); // Hello my name is Romain
```

ECMAScript 6 - Default Params

- Les paramètres d'entrées peuvent maintenant recevoir un valeur par défaut si rien ne leur est transmis
- La valeur par défaut peut être un appel à une fonction une création d'objet

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

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

var frDate = function(date = new Date()) {
  var day = (date.getDate() < 10) ? '0' + date.getDate() : date.getDate();
  var month = (date.getMonth() + 1 < 10) ? '0' + (date.getMonth() + 1) :
date.getMonth() + 1;
  var year = date.getFullYear();
  return day + '/' + month + '/' + year;
};

console.log(frDate(new Date('1985-10-01'))); // 01/10/1985
console.log(frDate()); // 14/11/2016
```

ECMAScript 6 - Rest Parameters

- Un Rest parameter permet de récupérer des arguments multiple dans une seul variable de type Array.

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

  c.forEach(n => result += n);

  return result;
};

var sum = function(...n) {
  return n.reduce((a, b) => a + b);
};

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

ECMAScript 6 - Spread Parameter

- Le Spread parameter permet de renseigner plusieurs arguments avec un seul paramètres de type Array

```
var sum = function(a, b, c, d) {
  return a + b + c + d;
};

var nbs = [1, 2, 3, 4];

console.log(sum(...nbs)); // 10
```

ECMAScript 6 - Boucle for..of

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

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

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

ECMAScript 6 - Symbol

- Symbol est un nouveau type primitif qui n'a pas de syntaxe litéral, seul l'appel à la fonction Symbol est possible
- 2 appel successif 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

Modules JavaScript

Modules JavaScript

- JavaScript inventé en 1995 par Netscape

  Objectif : créer des interactions côté client, après chargement de la page

  Exemple de l'époque :

  - Menu en rollover (image ou couleur change au survol)
  - Validation de formulaire

- JavaScript aujourd'hui
  - Permet la création d'application front-end, back-end, en ligne de commande, application de bureau
  - Ces applications peuvent contenir plusieurs centaines de milliers de lignes de codes
  - Il faut faciliter le travail collaboratif, en plusieurs fichiers et en limitant les risques de conflit

Modules JavaScript

- Immediately-invoked function expression (IIFE)

```
// jquery-button.js
(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));
```

- Une fonction expression anonyme appelée immédiatement
  - Limite la portée des variables
  - Permet de renommer localement des dépendances

Modules JavaScript

- Utilisation

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Exemple</title>
</head>
<body>
    <div id="container"></div>
    <script src="http://code.jquery.com/jquery-1.11.3.min.js"></script>
    <script src="jquery-button.js"></script>
    <script>
        var button = new MonBouton({
            value: 'Cliquez ici'
        });

        button.creer('#container');
    </script>
</body>
</html>
```

- Inconvénients
  - L’ordre d’inclusion des scripts doit être connu (ici jQuery avant jquery-button)
  - Les modules reçoivent leur dépendances via des variables globales (jQuery, window)
  - Les modules exposent leur code via des variables globales (global.MonBouton)

Modules JavaScript

- Modules YUI

  Yahoo User Interface library (plus maintenue depuis mi-2014)
  Première bibliothèque à introduire la notion de modules
  http://yuilibrary.com/yui/docs/yui/create.html

```
// yui-button.js
YUI().add('mon-bouton', function (Y) {
  'use strict';

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

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

  Y.MonBouton = MonBouton;
}, '0.0.1', {
  requires: ['node']
});
```

- Un module YUI décrit ses dépendances (requires: [‘node'] pour accéder aux méthodes on et append)
- Pas d’utilisation de variables globales

Modules JavaScript

- Utilisation

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Exemple</title>
</head>
<body>
    <div id="container"></div>
    <script src="http://yui.yahooapis.com/3.18.1/build/yui/yui-min.js"></script>
    <script>
        YUI({
            modules: {
                'mon-bouton': 'yui-button.js'
            }
        }).use('mon-bouton', function (Y) {
            var button = new Y.MonBouton({
                value: 'Cliquez ici'
            });

            button.creer('#container');
        });
    </script>
</body>
</html>
```

Modules JavaScript

- CommonJS

  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://www.commonjs.org/specs/modules/1.0/

- Création d'un module

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

  - Les modules commons JS exposent à l'intérieur d'un module une variable exports de type object (et qui peut être écrasée si besoin)

Modules JavaScript

- Utilisation

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

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

  - CommonJS propose une méthode require pour le chargement de modules, dont le retour correspond à la variable exports
  - Cependant CommonJS ne s’applique pas au navigateur où le chargement de fichiers se fait via la balise script

Modules JavaScript

- Browserify
  Permet de charger des modules CommonJS côté client.
- Installation :
  npm install -g browserify
- Transormation 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>
```

Modules JavaScript

- Asynchronous Module Definition
  CommonJS ne permettant pas d'exécuter de charger des modules côté client, AMD est né.
- RequireJS
  Plusieurs bibliothèques permettent de charger des modules AMD, RequireJS est la plus connue.
  http://requirejs.org/
- RequireJS définie 2 fonctions globales require et define. define permet de définir un module, require est le point d'entrée de l'application.

Modules JavaScript

```js
// number-converter.js
define(function() {
  var exports = {};

  exports.convert = function(nb) {
    return Number(nb);
  };

  return exports;
});
```

```js
// calculette.js
define(['number-converter'], function(numberConverter) {
  var exports = {};

  exports.ajouter = function(nb1, nb2) {
    return numberConverter.convert(nb1) + numberConverter.convert(nb2);
  };

  return exports;
});
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
    <script src="bower_components/requirejs/require.js"></script>
    <script>
        require(['calculette'], function(calc) {
            console.log(calc.ajouter(2, 3)); // 5
        });
    </script>
</body>
</html>
```

Modules JavaScript

- ECMAScript 2015 / ECMAScript 6
  La nouvelle version de JavaScript prévoit une syntaxe pour l'utilisation de module. A l'heure actuelle (juillet 2015), ni les navigateurs ni Node.js ou io.js ne supportent cette syntaxe.

- Babel / Traceur
  Babel et Traceur sont des bibliothèques qui permettent de transpiler du code ES6 en ES5 et ainsi l'utiliser sur les moteurs actuels.

- Installation :
  npm install -g babel

- Utilisation (toutes les sources du répertoires src vers le répertoire dist) :
  babel src --out-dir dist/

Modules JavaScript

```
// src/number-converter.js
var exports = {};

exports.convert = function(nb) {
  return Number(nb);
};

export default exports;
```

```
// src/calculette.js
import numberConverter from './number-converter';

var exports = {};

exports.ajouter = function(nb1, nb2) {
  return numberConverter.convert(nb1) + numberConverter.convert(nb2);
};

export default exports;
```

```
// src/main.js
import calc from './calculette';

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

Modules JavaScript

- ## Universal Module Definition

  L'objectif d'UMD est de proposer des modules compatibles CommonJS, AMD ou en utilisant des variables globales si le contexte ne permet pas d'utiliser les 2 précédents.
  https://github.com/umdjs/umd

```
// number-converter.js
(function (root, factory) {
  if (typeof exports === 'object') {
    // CommonJS
    module.exports = factory();
  } else if (typeof define === 'function' && define.amd) {
    // AMD
    define(function () {
      return (root.numberConverter = factory());
    });
  } else {
    // Global Variables
    root.numberConverter = factory();
  }
}(this, function () {
  var exports = {};

  exports.convert = function(nb) {
    return Number(nb);
  };

  return exports;
}));
```

```
// calculette.js
(function (root, factory) {
  if (typeof exports === 'object') {
    // CommonJS
    module.exports = factory(require('./number-converter'));
  } else if (typeof define === 'function' && define.amd) {
    // AMD
    define(['./number-converter'], function (numberConverter) {
      return (root.calculette = factory(numberConverter));
    });
  } else {
    // Global Variables
    root.calculette = factory(root.numberConverter);
  }
}(this, function (numberConverter) {
  var exports = {};

  exports.ajouter = function(nb1, nb2) {
    return numberConverter.convert(nb1) +
numberConverter.convert(nb2);
  };

  return exports;
}));
```

Modules JavaScript

- System.js
  System.js 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

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’une code au moment où une interaction se produit (ex : clic, fin de chargement, reception 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

- Alors que vaut this ?

  Tout dépend de ce qui ce passe dans l'API qui va appeler le callback.
  Dans Node.js le callback est appelé dans la fonction ontimeout :
  timer._onTimeout(), this est donc l'objet sur lequel est appelé le callback, ici timer qui est une instance de Timer.

```
// Version très simplifiée de setTimeout extraite de
// https://github.com/nodejs/node/blob/master/lib/timers.js
exports.setTimeout = function(callback, after, args) {
  var timer = new Timeout(after, callback, args);
  return timer;
};

function ontimeout(timer) {
  var args = timer._timerArgs;
  if (!args)
    timer._onTimeout();
}

function Timeout(after, callback, args) {
  this._idleTimeout = after;
  this._onTimeout = callback;
  this._timerArgs = args;
}
```

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: function() {
    setTimeout(() => {
      console.log('Hello my name is ' + this.firstName);
    }, 1000)
  }
};

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

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 grace à des bibliothèques comme bluebird ou q, ou bien nativement depuis ES6.

```
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.readFile(srcFilePath)
  .then(content => fsp.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

- Observables

  Les promesses ont leur limite, il faut recréer une promesse si elle se répète, il est également impossible de les annuler. Dans ce cas la tendance est aux bservables, via la bibliothèque rxJS et bientôt intégrés au langage. On parle de Reactive Programming

- Angular 2 intègre rxJS par défaut

JavaScript Asynchrone - Exercice

- Remplacer les appels à Mongoose par l'utilisation de Promesse
- Utiliser les méthodes then et catch

formation.tech

Gestion de dépendances

Gestion de dépendances

- Gestionnaire de dépendance de node.js (s'installe en même temps que node)
- Equivalent pour du code JavaScript à apt-get
- Plutôt destiné à du code console ou serveur, bien que des bibliothèques comme jQuery ou Bootstrap y soient présentes

Gestion de dépendances

- Trouver des packages
  https://www.npmjs.com
- Créer un package
  npm init
- Le fichier package.json
  http://browsenpm.org/package.json

Gestion de dépendances

- Installer un package

  npm install <package>
  npm install <package> --save
  npm install <package>@<version> --save

  Ex : npm install jquery@1.11.*

- Mettre à jour les packages installés

  npm update
  npm update --save
  npm update <package>
  npm update <package> --save
  npm update <package>@<version> --save

- Désinstaller

  npm uninstall lodash
  npm uninstall lodash --save

Gestion de dépendances

- 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
```

- Verrouiller des dépendances

```
npm shrinkwrap
```

- Détecter des dépendances plus à jour

```
npm outdated
```

Gestion de dépendances

- Bower
  Gestionnaire de dépendance pour bibliothèques front-end (CSS/JS/Polices…). Créé par Twitter en 2012
- Pré-requis
  Node.js
  Git
- Installation
  npm install -g bower
- Créer un projet
  bower init
- Trouver des packages
  http://bower.io/search/

Gestion de dépendances

- Installer un package
  bower install <package>
  bower install <package>#<version>

  Ex : bower install jquery#1.11.*

- Mettre à jour
  bower update

- Configuration
  Fichier .bowerrc
  http://bower.io/docs/config/

- Dépôts privés :
  https://github.com/bower/registry

Node.js

Node.js - Introduction

- Créé 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

```js
/* Un simple helloword */

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

setInterval(helloworld, 1000);
```

- Lancement du programme

  node FILE_PATH.js

- Interruption

  CTRL-C

Node.js - Pourquoi JavaScript côté serveur ?

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

http.createServer(function(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 - Module HTTP

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

http.createServer(function(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 - Modules

- Node.js contient un certain nombre de modules prédéfinis
  https://nodejs.org/api/

- Assertion Testing
- Buffer
- C/C++ Addons
- Child Processes
- Cluster
- Console
- Crypto
- Debugger
- DNS
- Domain
- Errors
- Events
- File System
- Globals
- HTTP
- HTTPS
- Modules
- Net
- OS
- Path
- Process
- Punycode
- Query Strings
- Readline
- REPL
- Stream
- String Decoder
- Timers
- TLS/SSL
- TTY
- UDP/Datagram
- URL
- Utilities
- V8
- VM
- ZLIB

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 permette de passer un id quelconque (utiliser une expression régulière ou bien la méthode substr du type String)

Grunt

Grunt

- Grunt JS

  Permet l'automatisation de tâches de développement front-end.

- Exemples
  - minifier ses fichiers JS
  - compiler ses CSS
  - compresser les images
  - exécuter les tests
  - vérifier les conventions de codage

Grunt

- Installation via npm :

```
npm install -g grunt-cli
```

Gruntfile.js

```
/*global module:false*/
module.exports = function(grunt) {

  grunt.initConfig({
    copy: {
      dist: {
        src: 'index.html',
        dest: 'dist/index.html'
      }
    },
    uglify: {
      dist: {
        src: 'script.js',
        dest: 'dist/script.js'
      }
    }
  });

  grunt.loadNpmTasks('grunt-contrib-copy');
  grunt.loadNpmTasks('grunt-contrib-uglify');

  // Default task.
  grunt.registerTask('default', ['copy',
'uglify']);

};
```

package.json

```
{
  "engines": {
    "node": ">= 0.10.0"
  },
  "devDependencies": {
    "grunt": "^0.4.5",
    "grunt-contrib-copy": "^0.8.0",
    "grunt-contrib-uglify": "^0.9.1"
  }
}
```

Grunt

src/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
    <div>
        Prénom : <input type="text" id="prenom">
    </div>
    <p>
        Bonjour <span id="output"></span>
    </p>
    <script src="script.js"></script>
</body>
</html>
```

copy

→

dist/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
    <div>
        Prénom : <input type="text" id="prenom">
    </div>
    <p>
        Bonjour <span id="output"></span>
    </p>
    <script src="script.js"></script>
</body>
</html>
```

src/script.js

```
!function() {
    'use strict';

    var inputElt =
document.querySelector('#prenom');
    var outputElt =
document.querySelector('#output');

    inputElt.addEventListener('input', function()
{
        outputElt.innerHTML = inputElt.value;
    });
}();
```

uglify

→

dist/script.js

```
!function(){"use strict";var
a=document.querySelector("#prenom"),b=document.qu
erySelector("#output");a.addEventListener("input"
,function(){b.innerHTML=a.value})}();
```

Grunt

Gruntfile.js

```
/*global module:false*/
module.exports = function(grunt) {

  grunt.initConfig({
    copy: {
      dist: {
        src: 'index.html',
        dest: 'dist/index.html'
      }
    },
    uglify: {
      dist: {
        src: 'script.js',
        dest: 'dist/script.js'
      }
    }
  });

  grunt.loadNpmTasks('grunt-contrib-copy');
  grunt.loadNpmTasks('grunt-contrib-uglify');

  // Default task.
  grunt.registerTask('default', ['copy', 'uglify']);

};
```

package.json

```
{
  "devDependencies": {
    "grunt": "^0.4.5",
    "grunt-contrib-copy": "^0.8.0",
    "grunt-contrib-uglify": "^0.9.1"
  }
}
```

- package.json créé avec :
  npm init
  npm install grunt --save-dev
  npm install grunt-contrib-copy --save-dev
  npm install grunt-contrib-uglify --save-dev

Grunt

- Liste des plugins pour grunt :
  http://gruntjs.com/plugins
  (4,403 plugins en juillet 2015)
- Les plugins contrib-* sont ceux des développeurs de grunt.

Grunt

- jit-grunt :

  Installation : `npm install jit-grunt --save-dev`

  Simplifie le chargement de plugins

Avant

```
/*global module:false*/
module.exports = function(grunt) {

  grunt.loadNpmTasks('grunt-contrib-clean');
  grunt.loadNpmTasks('grunt-contrib-concat');
  grunt.loadNpmTasks('grunt-contrib-copy');
  grunt.loadNpmTasks('grunt-contrib-cssmin');
  grunt.loadNpmTasks('grunt-contrib-jshint');
  grunt.loadNpmTasks('grunt-contrib-less');
  grunt.loadNpmTasks('grunt-contrib-uglify');
  grunt.loadNpmTasks('grunt-contrib-watch');
  grunt.loadNpmTasks('grunt-google-cdn');
  grunt.loadNpmTasks('grunt-rev');
  grunt.loadNpmTasks('grunt-spritesmith');
  grunt.loadNpmTasks('grunt-usemin');

  grunt.initConfig({
    // ...
  });

  // Default task.
  grunt.registerTask('default', [
    // ...
  ]);

};
```

Après

```
/*global module:false, require*/
module.exports = function(grunt) {
  'use strict';

  require('jit-grunt')(grunt, {
    useminPrepare: 'grunt-usemin',
    cdnify: 'grunt-google-cdn',
    sprite: 'grunt-spritesmith'
  });

  // Project configuration.
  grunt.initConfig({
    // ...
  });

  // Default task.
  grunt.registerTask('default', [
    // ...
  ]);

};
```

Grunt

- grunt-contrib-less :

  `npm install grunt-contrib-less --save-dev`

  Compile des fichiers LESS en CSS

```
module.exports = function(grunt) {

  // ...

  grunt.initConfig({
    less: {
      dev: {
        files: [{
          expand: true,
          cwd: 'less',
          src: ['*.less'],
          dest: 'css/',
          ext: '.css'
        }]
      }
    },
  });

  // Default task.
  grunt.registerTask('default', [
    // ...
  ]);

};
```

Grunt

- grunt-autoprefixer :

  `npm install grunt-autoprefixer --save-dev`

  Rajoute automatiquement les préfixes -moz, -webkit, -o, -ms en fonction des versions minimales des navigateurs à supporter

```
module.exports = function(grunt) {

  // ...

  grunt.initConfig({
    // ...
    autoprefixer: {
      options: {
        browsers: ['last 2 versions', 'ie 8', 'ie 9']
      },
      dev: {
        files: [{
          expand: true,
          cwd: 'css/',
          src: '{,*/}*.css',
          dest: 'css/'
        }]
      },
    },
  });

  // Default task.
  grunt.registerTask('default', [
    // ...
  ]);

};
```

Grunt

- grunt-contrib-watch :

  `npm install grunt-contrib-watch --save-dev`

  Surveille les modifications sur des fichiers, exécute des taches en cas de changement

```
module.exports = function(grunt) {

  // ...

  grunt.initConfig({
    // ...
    watch: {
      less: {
        files: ['less/**/*.less'],
        tasks: ['less:dev', 'autoprefixer:dev']
      }
    },
  });

  // Default task.
  grunt.registerTask('default', [
    // ...
  ]);

};
```

Grunt

- grunt-contrib-concat :
  `npm install grunt-contrib-concat --save-dev`
  Concatène plusieurs fichiers en un. Utile pour optimiser les temps de chargement CSS/JS
- grunt-contrib-uglify :
  `npm install grunt-contrib-uglify --save-dev`
  Compresse les fichiers JS
- grunt-contrib-cssmin :
  `npm install grunt-contrib-cssmin --save-dev`
  Compresse les fichiers CSS

Grunt

- grunt-contrib-copy :
  `npm install grunt-contrib-copy --save-dev`
  Copie des fichiers
- grunt-contrib-clean :
  `npm install grunt-contrib-clean --save-dev`
  Supprime des fichiers

Grunt

- grunt-usemin:

  `npm install grunt-usemin --save-dev`

  Génère une configuration pour concat, uglify, cssmin à partir d'un fichier HTML

index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title></title>

    <!-- build:css css/app.css -->
    <link rel="stylesheet" href="css/body.css">
    <link rel="stylesheet" href="css/button.css">
    <!-- endbuild -->
</head>
<body>

<!-- build:js js/app.js -->
<script src="js/create-button.js"></script>
<script src="js/button-listener.js"></script>
<!-- endbuild -->
</body>
</html>
```

Gruntfile.js

```js
/*global module, require*/
module.exports = function(grunt) {
  'use strict';

  // ...

  grunt.initConfig({
    // ...
    useminPrepare: {
      html: 'index.html'
    },

    usemin: {
      html: ['dist/{,*/}*.html'],
      css: ['dist/{,*/}*.css'],
      js: ['dist/{,*/}*.js'],
    },

  });

  // Default task.
  grunt.registerTask('default', [
    // ...
  ]);
};
```

Grunt

config générée

```
{
  "concat": {
    "generated": {
      "files": [{
          "dest": ".tmp/concat/css/app.css",
          "src": ["css/body.css", "css/button.css"]
        }, {
          "dest": ".tmp/concat/js/app.js",
          "src": ["js/create-button.js", "js/button-
listener.js"]
      }]
    }
  },
  "uglify": {
    "generated": {
      "files": [{
          "dest": "dist/js/app.js",
          "src": [".tmp/concat/js/app.js"]
      }]
    }
  },
  "cssmin": {
    "generated": {
      "files": [{
          "dest": "dist/css/app.css",
          "src": [".tmp/concat/css/app.css"]
      }]
    }
  }
}
```

index.html généré

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title></title>

    <link rel="stylesheet" href="css/app.css">
</head>
<body>

<script src="js/app.js"></script>
</body>
</html>
```

app.css généré

```
body{background:beige}button{width:50px;height:50px}
```

app.js généré

```
!function(){"use strict";var
a=document.createElement("button");a.innerHTML=0,a.id="
monBouton",document.body.appendChild(a)}(),!function()
{"use strict";var
a=document.querySelector("#monBouton");a.addEventListen
er("click",function(){this.innerHTML++})}();
```

Grunt

- contrib-connect :
  serveur web
- karma :
  lancer des tests
- concurrent :
  exécuter des taches en parallèle
- sass :
  compile des fichiers SASS en CSS
- contrib-imagemin :
  compresser des images
- contrib-htmlmin :
  minifier le HTML
- newer :
  ne lancer les taches que sur les nouveaux fichiers
- rev :
  genère un nom de fichier avec hash pour le cache (avec usemin)
- contrib-jshint, jscs :
  vérifie les conventions sur les fichiers JS
- google-cdn :
  remplace les fichiers locaux par des CDN
- spritesmith :
  génère des fichiers Sprite CSS

Grunt

- Grunt Init
  Assistant de création de projet grunt

- Installation
  npm install -g grunt-init

- Création du projet
  grunt-init gruntfile

```
Please answer the following:
[?] Is the DOM involved in ANY way? (Y/n) Y
[?] Will files be concatenated or minified? (Y/n) Y
[?] Will you have a package.json file? (Y/n) Y
[?] Do you need to make any changes to the above before continuing? (y/N) N

Writing Gruntfile.js...OK
Writing package.json...OK

Initialized from template "gruntfile".
```

- Créer son propre assistant/plugin :
  https://github.com/gruntjs/grunt-init-gruntplugin

Grunt

- Gulp
  Equivalent de grunt, repose sur les streams Node.js (utilise la RAM plutôt que les fichiers).
  Devient très populaire, 1645 plugins contre 4403 pour grunt (juillet 2015)
- Broccoli
  484 plugins
- Brunch
  262 plugins
- Prepros / CodeKit
  https://prepros.io
  https://incident57.com/codekit/

gulpfile.js

```
var gulp = require('gulp');
var uglify = require('gulp-uglify');

gulp.task('scripts', function() {
  // Minify and copy all JavaScript (except vendor
scripts)
  gulp.src(['client/js/**/*.js', '!client/js/vendor/
**'])
    .pipe(uglify())
    .pipe(gulp.dest('build/js'));

  // Copy vendor files
  gulp.src('client/js/vendor/**')
    .pipe(gulp.dest('build/js/vendor'));
});

// The default task (called when you run `gulp`)
gulp.task('default', function() {
  gulp.run('scripts');

  // Watch files and run tasks if they change
  gulp.watch('client/js/**', function(event) {
    gulp.run('scripts');
  });
});
```

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)…

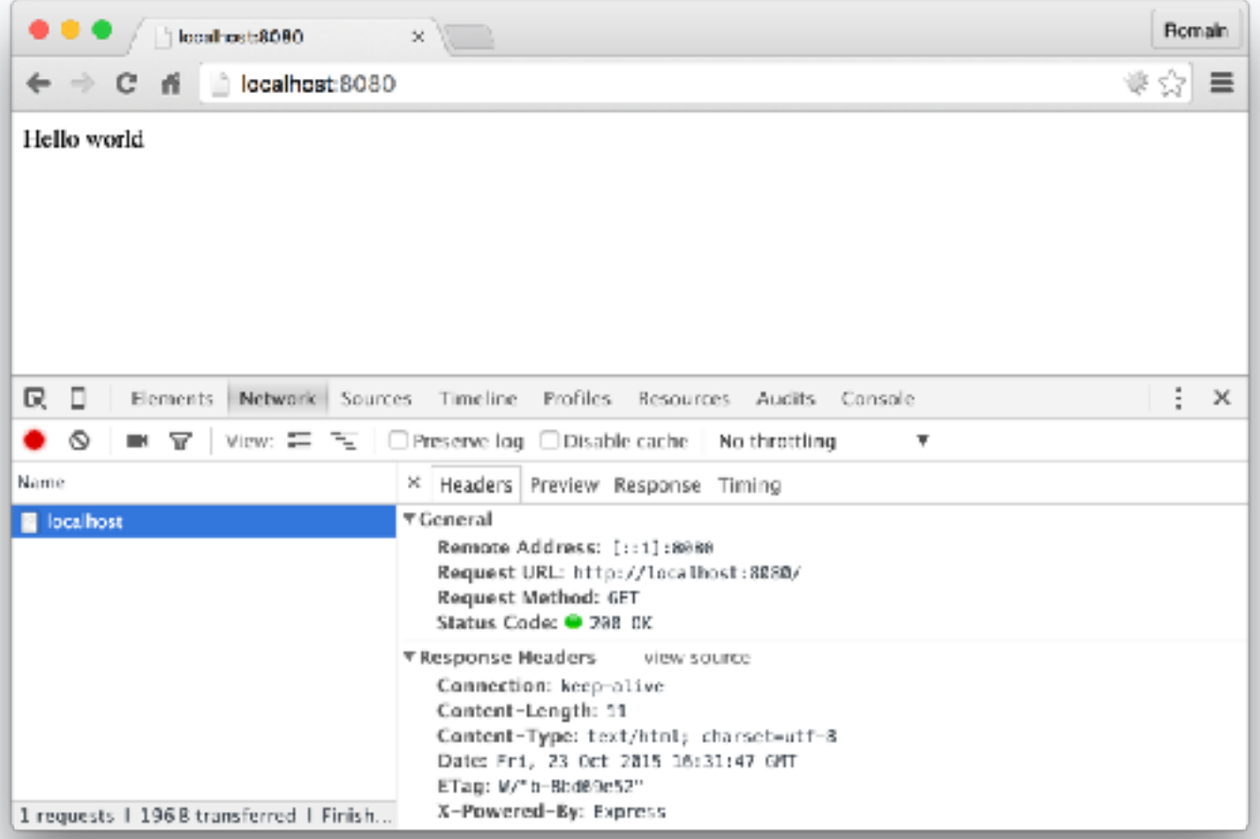
Express - Introduction

- 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/");
```



Express - MVC

- Définition

  L'architecture MVC est un Design Pattern apparu en Smalltalk et très répandu dans les frameworks web

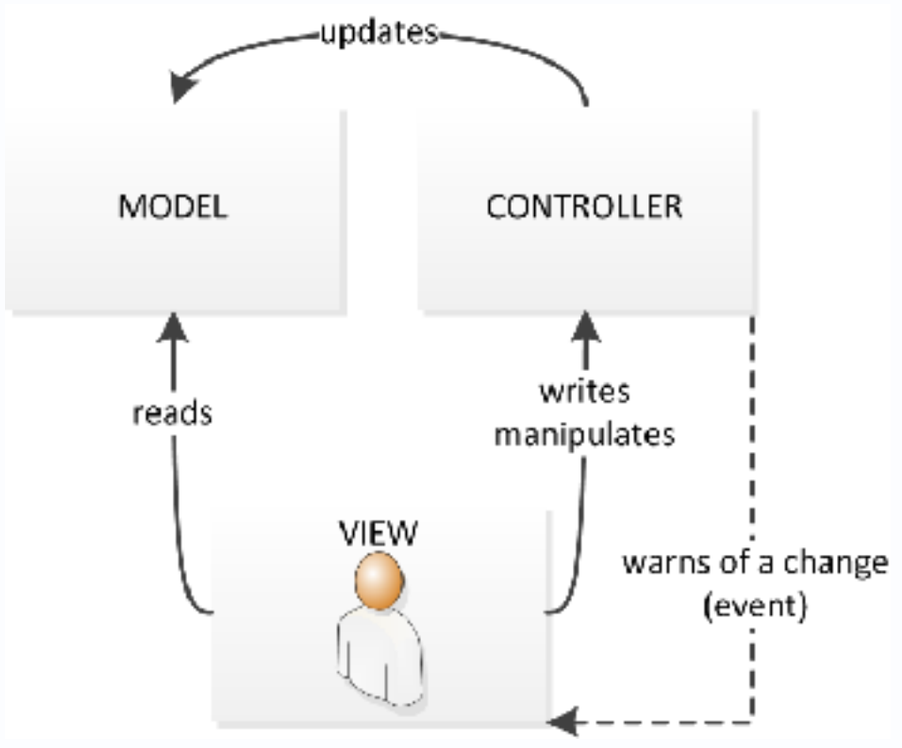
- Objectif

  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 : afficher ou non message d'erreur, menu qui dépend d'une authentification)
  - boucles foreach (uniquement foreach, ce qui impose d'avoir trié/filtré au préalable)
  - 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 fourni 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

```
Helloworld — -bash — 106×16
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'env PORT ou 3000)
  - 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 templates
  - Définition des middlewares à appeler avant le contrôleur
  - Définition des contrôleurs sur un préfixe d'URL
  - Middleware pour les erreurs 404
  - Middleware pour afficher les erreurs (avec env de dev et de prod)

```
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 balise h1 ayant comme contenu la variable title
  - Création de la balise p qui concatène Welcome to et la variable title

```
// 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

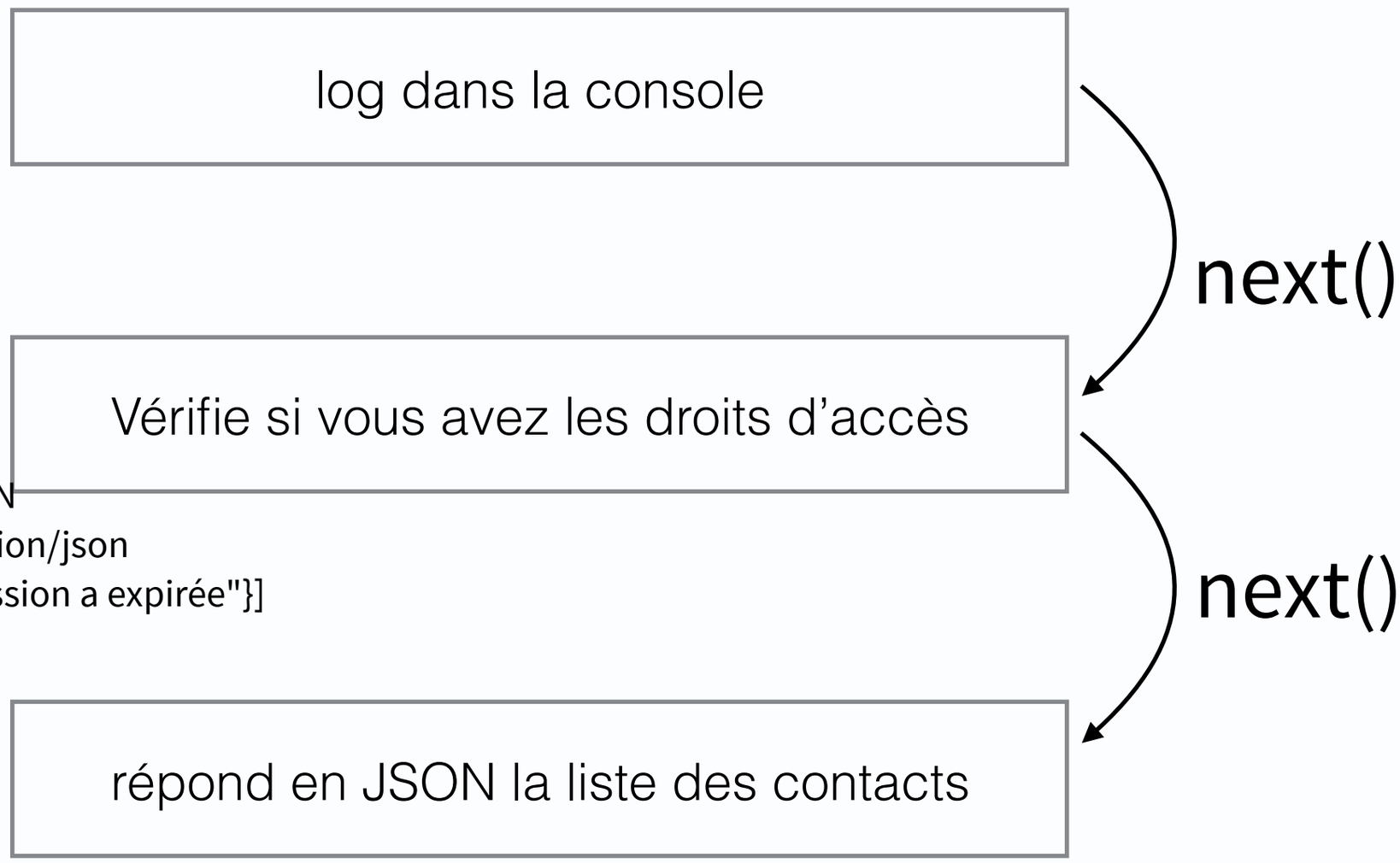
```
// GET /search?q=tobi+ferret
req.query.q
// => "tobi ferret"
```

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();
});
```

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 parsé

```
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);
```

```
{ fieldname: '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

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

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.

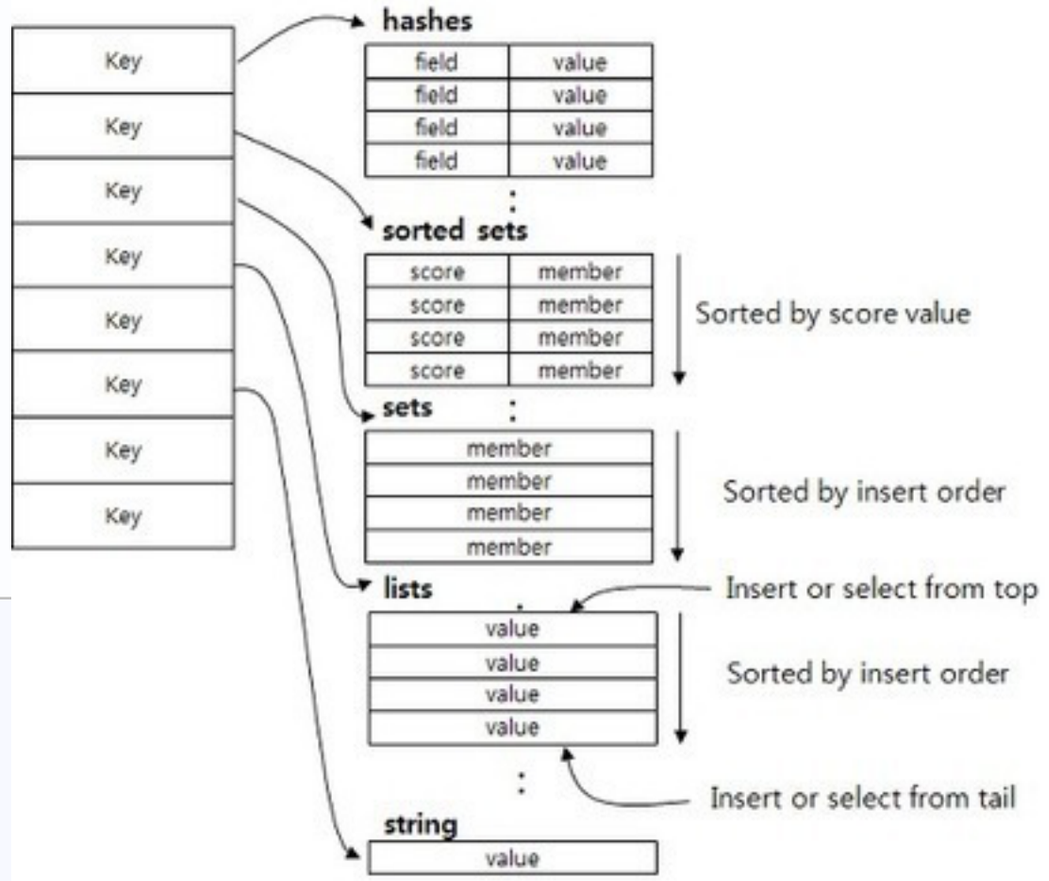
- Intérêts

  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();
});
```

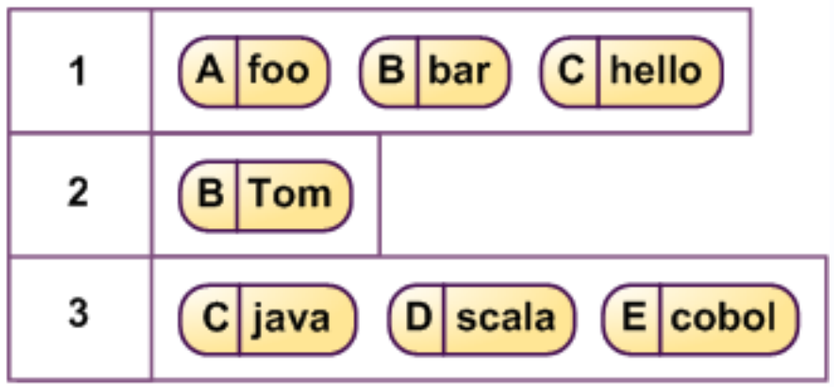
NoSQL - Orienté Colonne

- 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
    }
});
```

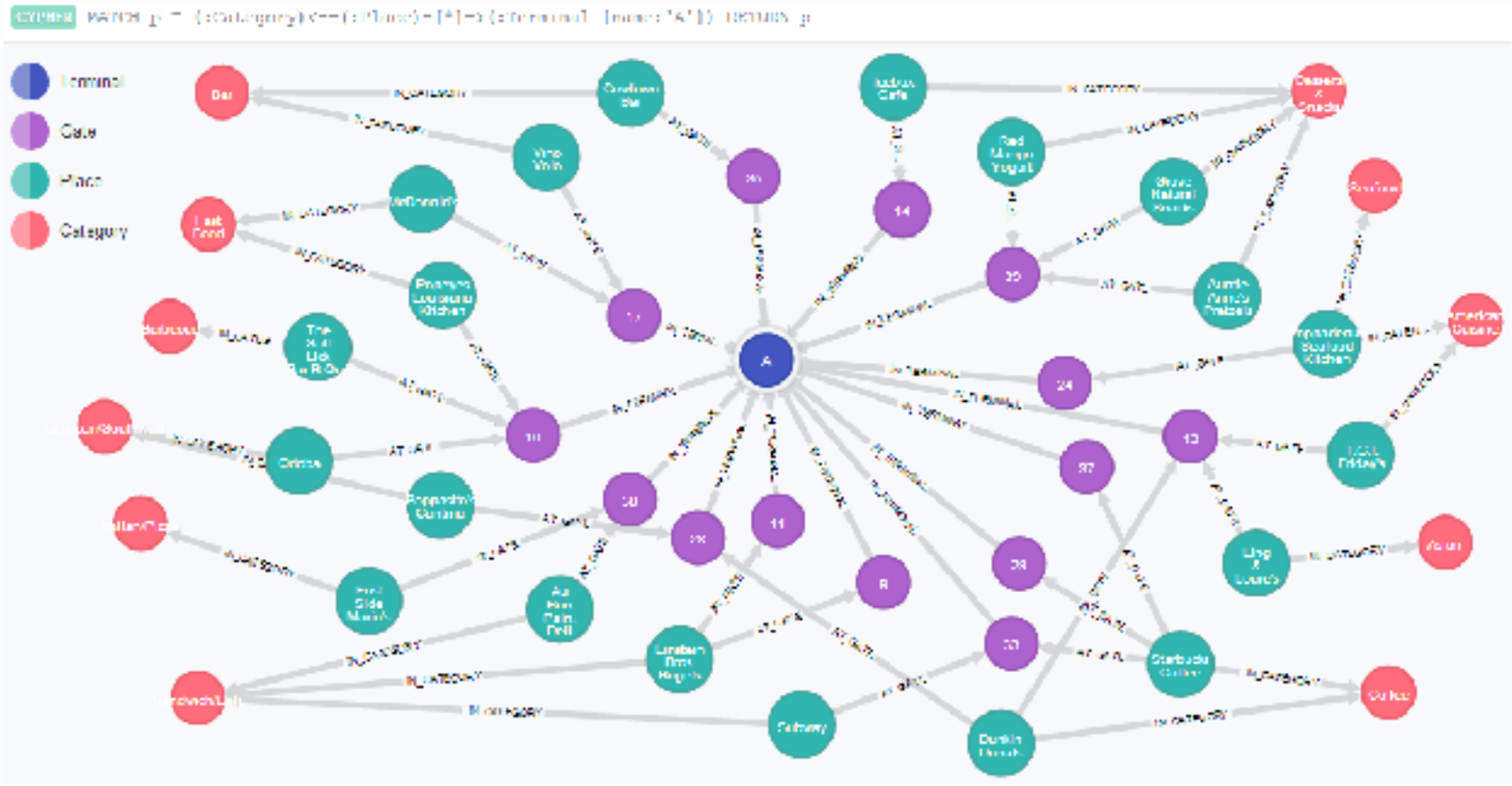
```
db.users.insert (      ← collection
   {
     name: "sue",      ← field: value
     age: 26,          ← field: value
     status: "A"       ← field: value
   }
)
```

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)); }
);
```



NoSQL - MongoDB

- ## 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

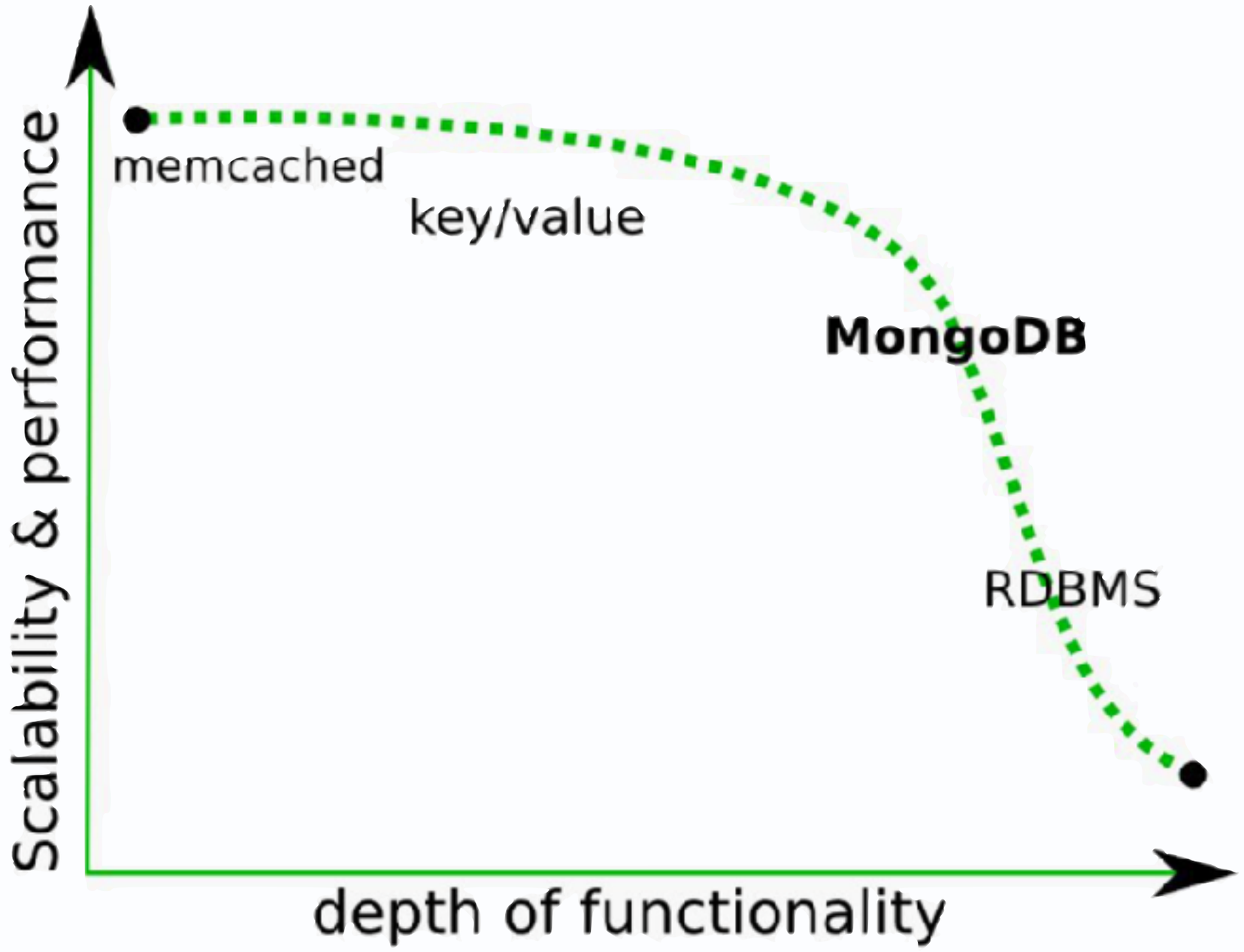
    https://www.mongodb.com/

    https://www.mongodb.org/dl/win32
  - Mac

    https://www.mongodb.com/

```
brew install mongo
```

NoSQL - MongoDB



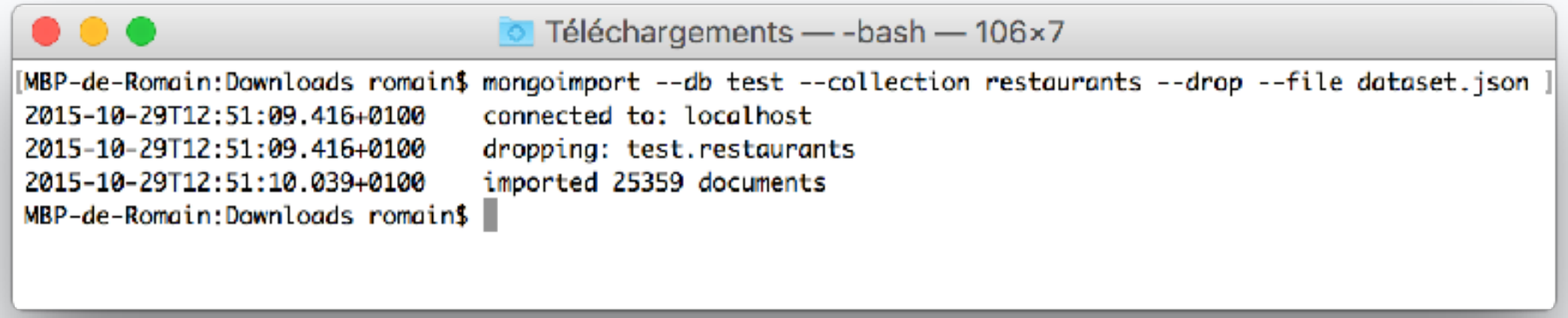
NoSQL - MongoDB

- 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 --db test --collection restaurants --drop --file ~/downloads/primer-dataset.json`



NoSQL - MongoDB

- MongoShell

  Mongo livre un programme client en ligne de commande pour accéder à la base.

```
romain — mongo — 91×17
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("562d4e918561c01ec2e43cfc"), "prenom" : "Bill", "nom" : "Gates" }
{ "_id" : ObjectId("562d4eab8561c01ec2e43cfd"), "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();
    });
});
```

```
MongoClient — -bash — 70×9
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

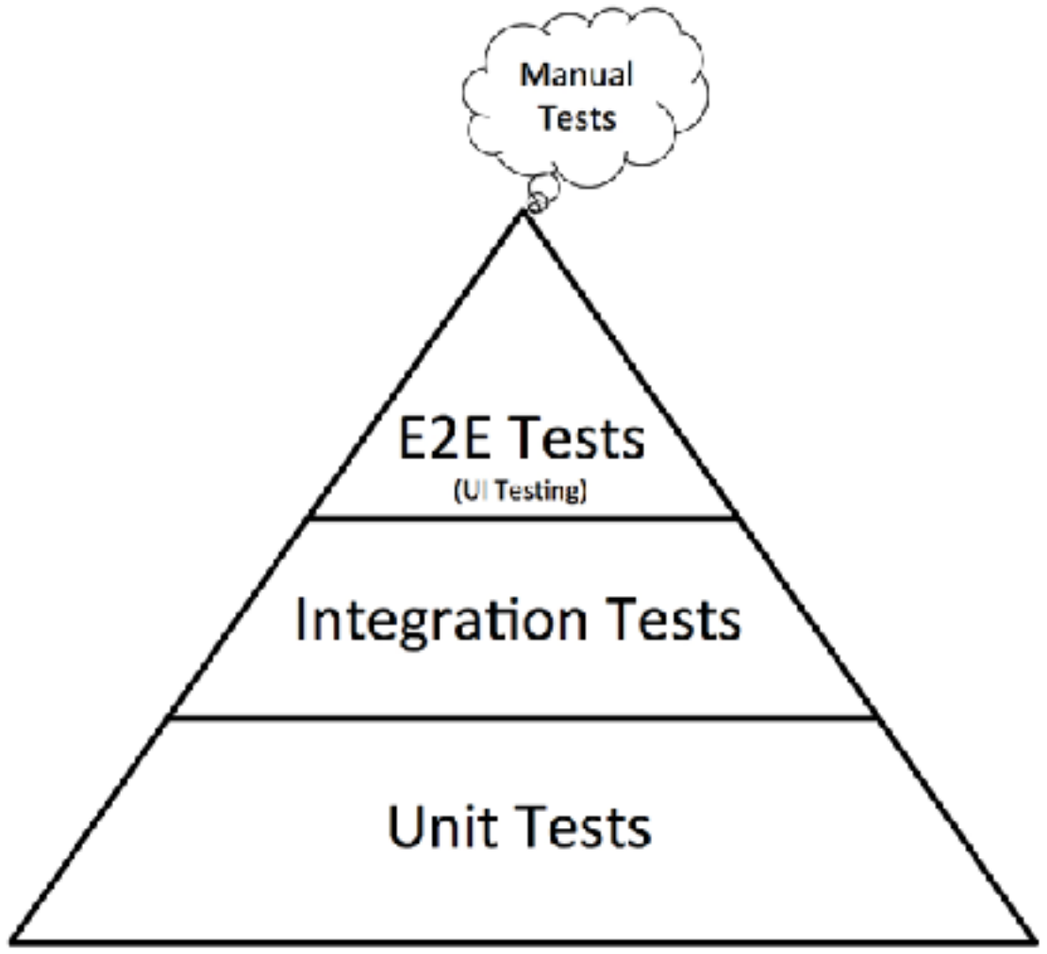
- Vérification manuelle
  - Ecrire une recette de tests et demander à une personne de la rejouer à des étapes clés (nouvelle version)
  - Ecrire le test sous la forme de code, et vérifier visuellement que les résultats attendus soit les bons

- 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

```
Air-de-Romain:Jasmine romains 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.
>
```

- Lancement des tests
  karma start

```
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 HE38slHTBKXL5t5yAAAA 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

- Créé 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

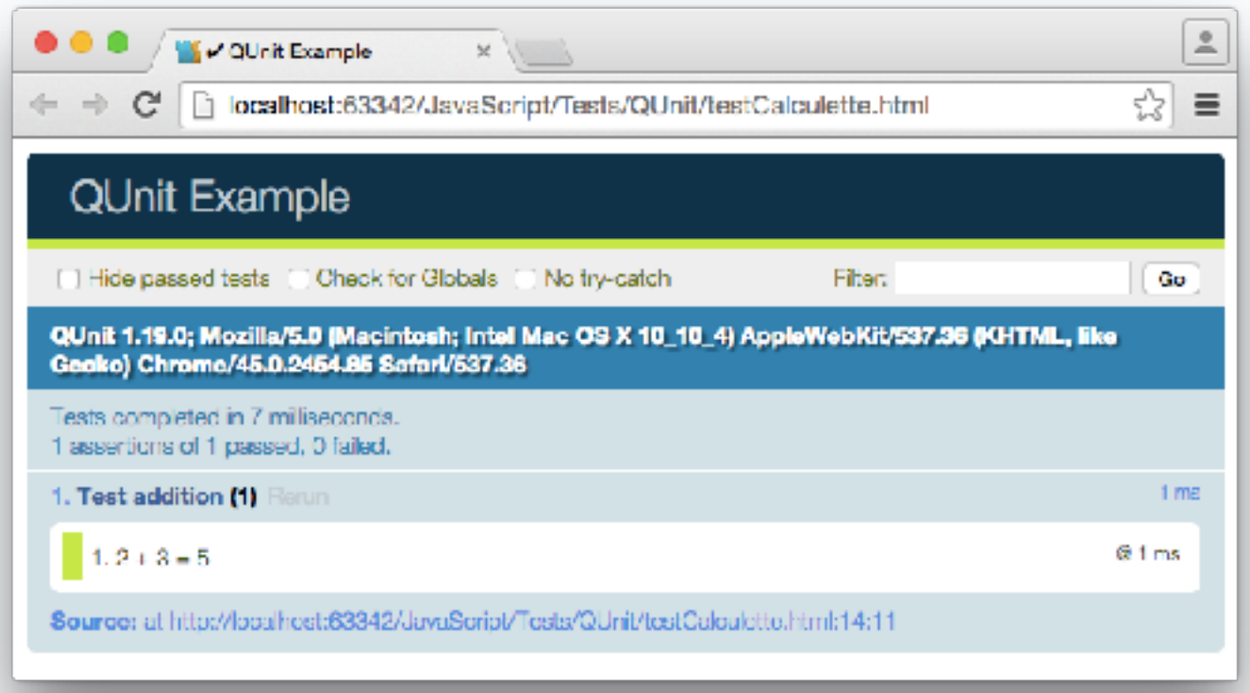
  grunt-contrib-qunit

  karma-qunit

Tests automatisés - QUnit

```html
<!-- 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>
```

```javascript
// calculette-test.js
QUnit.test("Test addition", function(assert) {
    assert.equal(calculette.ajouter(2, 3), 5, "2 + 3 = 5");
});
```

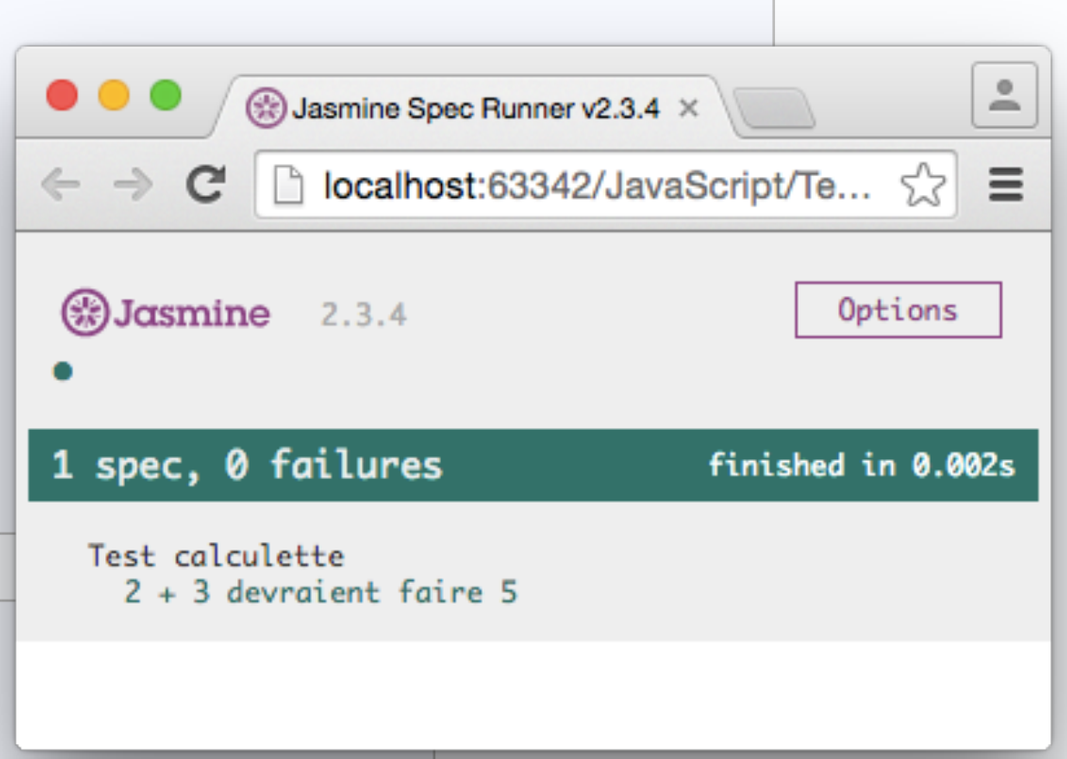


Tests automatisés - Jasmine

- Créé 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

- Créé 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

```html
<!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/calculette.js"></script>
    <script src="test/calculette-test.js"></script>
    <script>
      mocha.run();
    </script>
  </body>
</html>
```

```javascript
var assert = chai.assert;

describe('Test Addition', function() {
    it('2 + 3 devraient faire 5', function () {
        assert.equal(5, calculette.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