



**formation.tech**

# Formation Solutions Open Source, développement Front End

Romain Bohdanowicz

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

<http://formation.tech/>



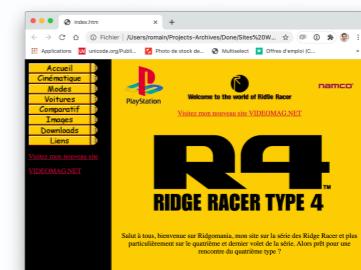
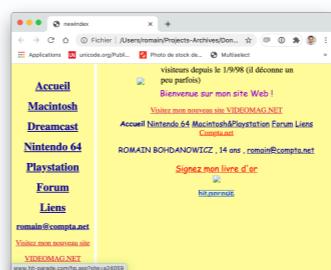
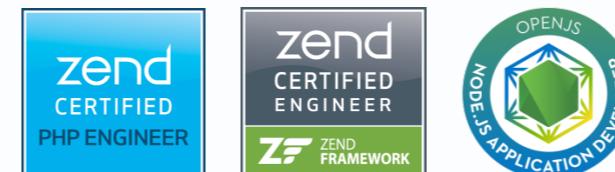
**formation.tech**

# Introduction



# Introduction - Formateur

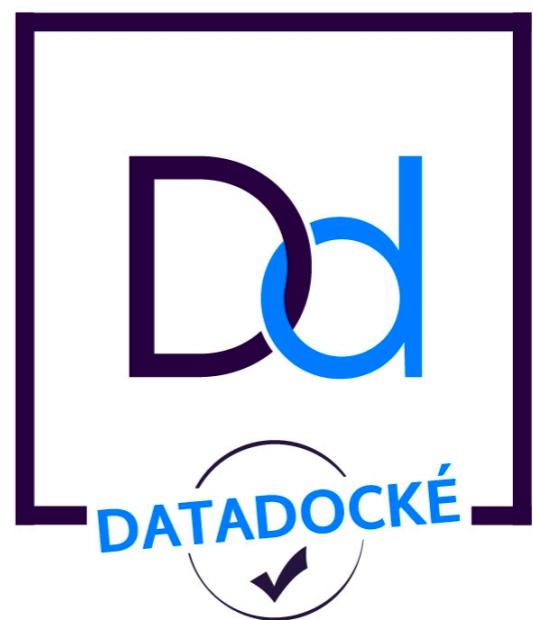
- Romain Bohdanowicz  
Ingénieur EFREI 2008, spécialité en Ingénierie Logicielle
- Expérience  
Formateur/Développeur Freelance depuis 2006  
Près de 2000 jours de formation animées
- Langages  
Expert : HTML / CSS / JavaScript / TypeScript / PHP / Java  
Notions : C / C++ / Objective-C / C# / Python / Bash / Batch
- Certifications  
PHP / Zend Framework / Node.js
- A propos  
Premier site web à 12 ans (HTML/JS/PHP)  
Triathlète du dimanche



# Introduction - formation.tech



- Organisme de formation depuis 2016
- Référencé DataDock
- Certifié Qualiopi
- 15 formations au catalogue
- Une dizaine de formateurs indépendants
- Formations en français ou anglais
- <https://formation.tech/>



# Introduction - WeAreDevs



- Studio de développement créé en 2017
- 1 salarié développeur senior
- Principales références
  - Cinexpert / Adeum
  - Sponsorise.me
  - Intel
  - Staytuned
  - STMicroelectronics
  - Miameur
  - Safran
- <https://wearedevs.fr/>



# Introduction - Et vous ?



- Pré-requis ?
- Rôle dans votre société ?
- Intérêt / objectif de cette formation ?



**formation.tech**

# JavaScript IDEs



# JavaScript IDEs - Webstorm

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





# JavaScript IDEs - Webstorm

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

Language > Array > functionnal.js

Project Structure

Language ~ /www/Learning/JavaScript/Language

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

Run functionnal.js

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

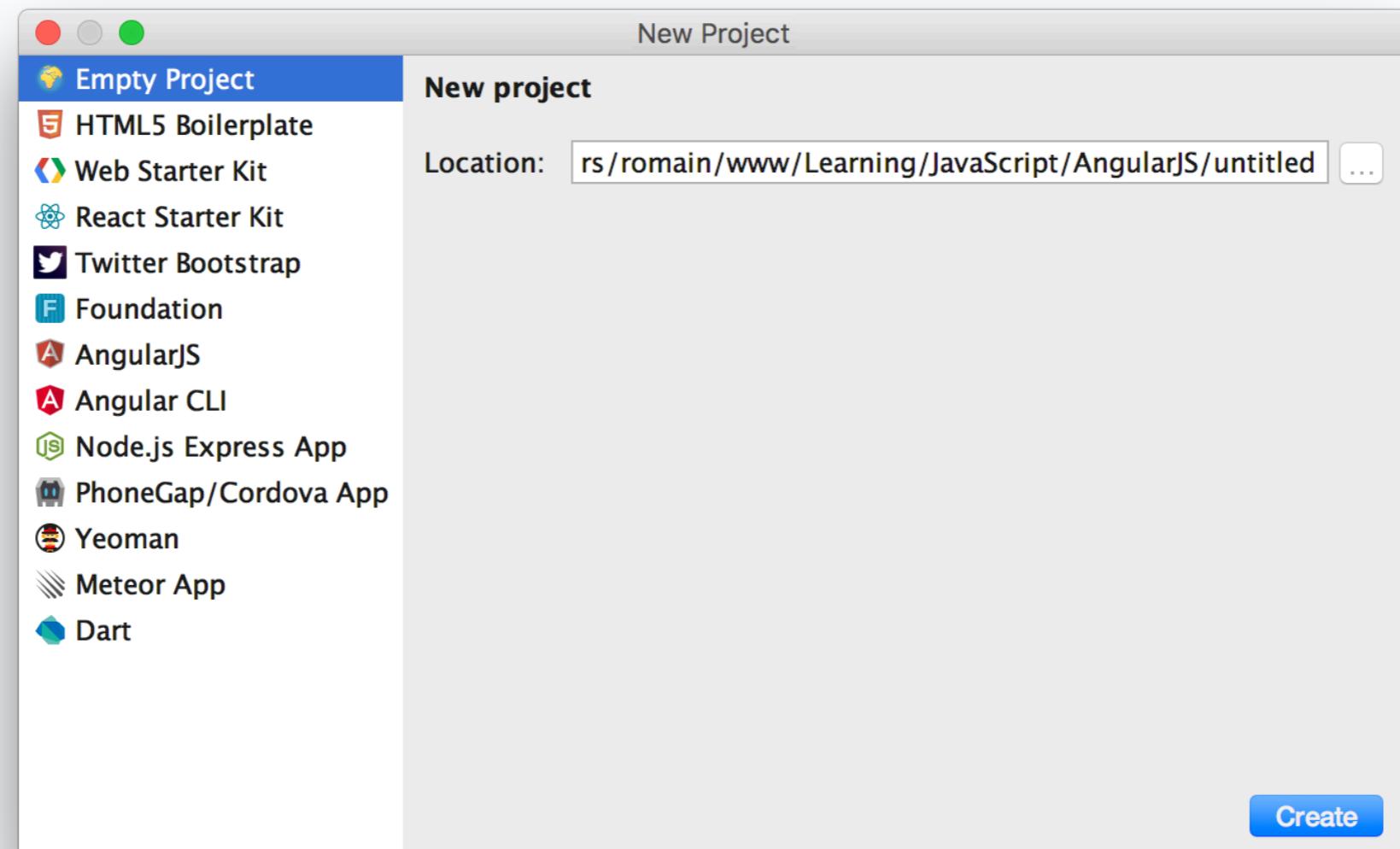
Process finished with exit code 0

4: Run 6: TODO Terminal Event Log

11:1 LF UTF-8

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

# JavaScript IDEs - Webstorm



# JavaScript IDEs - Webstorm



Run/Debug Configurations

Configuration    Browser / Live Edit    V8 Profiling

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

Node parameters:

Working directory:

JavaScript file:

Application parameters:

Environment variables:

▼ Before launch: Activate tool window

There are no tasks to run before launch

+ - ⚪ ▲ ▼

Show this page  Activate tool window

Cancel    Apply    OK

The screenshot shows the 'Run/Debug Configurations' dialog in Webstorm. The left sidebar lists various configuration types under 'Node.js' and 'Defaults'. The 'Node.js' section is expanded, showing options like Node.js Remote Debug, Nodeunit, PhoneGap/Cordova, Spy-js, Spy-js for Node.js, XSLT, and npm. The 'Configuration' tab is selected. The 'Node interpreter' field is set to '/usr/local/bin/node (Project)' with version 7.0.0. Below it are fields for 'Node parameters', 'Working directory', 'JavaScript file', 'Application parameters', and 'Environment variables'. A section titled 'Before launch: Activate tool window' indicates 'There are no tasks to run before launch'. At the bottom, there are checkboxes for 'Show this page' and 'Activate tool window', along with standard 'Cancel', 'Apply', and 'OK' buttons.

# JavaScript IDEs - Webstorm



Preferences

Languages & Frameworks > JavaScript > Libraries For current project

Libraries

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

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

Cancel Apply OK

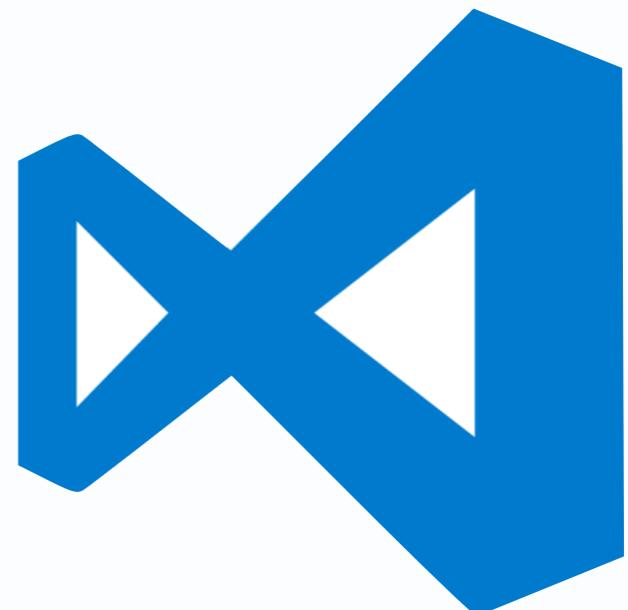
Search

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

# JavaScript IDEs - Visual Studio Code



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



# JavaScript IDEs - Visual Studio Code



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

- EXPLORATEUR (Left Sidebar):** Shows the project structure with files like `about.module.ts`, `parse5-adapter.js`, `api.ts`, `app.node.module.ts`, `app.browser.module.ts`, `home.module.ts`, `index.js`, `client.ts`, and `index.html`.
- EDITOR (Main Area):** Displays the `about.module.ts` file content:

```
1 import { Title } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3
4 import { AboutComponent } from './about.component';
5 import { AboutRoutingModule } from './about-routing.module';
6
7 @NgModule({
8   imports: [
9     AboutRoutingModule
10 ],
11 declarations: [
12   AboutComponent
13 ],
14 providers: [
15   Title
16 ],
17 })
18 export class AboutModule { }
```
- CONSOLE DE DÉBOGAGE (Bottom):** An empty debug console.
- STATUS BAR (Bottom):** Shows the current branch is `master*`, 29 lines of code, 2 errors, 0 warnings, and the file has 14 lines of code.
- STATUS BAR (Bottom Right):** Shows the file is 19 columns wide, 2 spaces, using LF line endings, is a TypeScript file, and includes a smiley face icon.



# JavaScript IDEs - Visual Studio Code

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

- Title Bar:** hello.js - VisualStudioCode
- Editor:** JS hello.js (line 3 is highlighted in yellow)
- Variables View:** Local variable message is set to "Hello".
- Call Stack View:** EN PAUSE SUR POINT D'ARRÊT (Breakpoint Hit). The stack shows:
  - hello at hello.js:3
  - ontimeout at timers.js:365
  - tryOnTimeout at timers.js:237
  - listOnTimeout at timers.js:207
- Breakpoints View:** Points d'Arrêt (Breakpoints) section with checkboxes for "Toutes les exceptions", "Exceptions interceptées" (checked), and "hello.js" (checked).
- Output Panel:** SORTIE (Output) tab showing ESLint output:

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

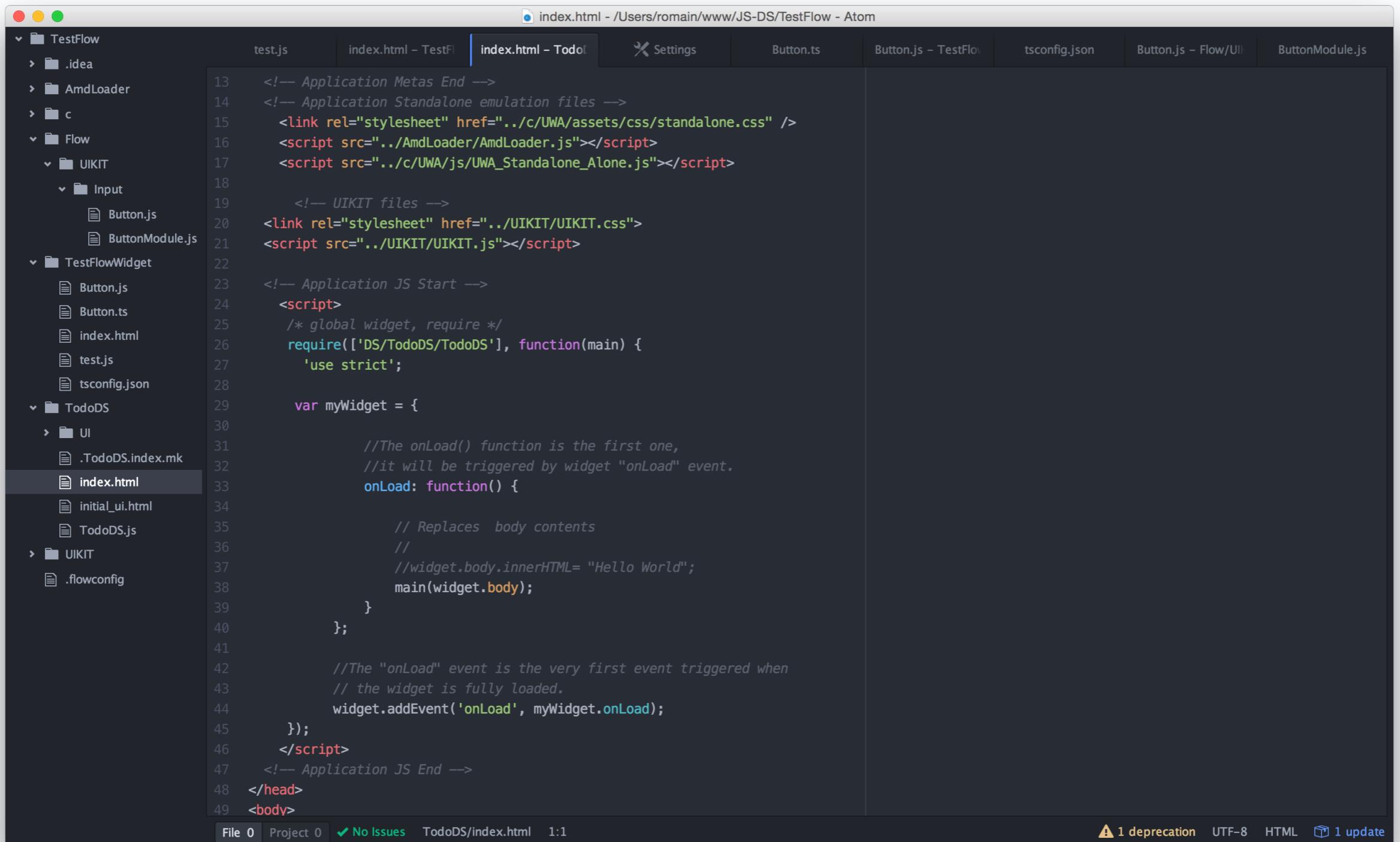
# JavaScript IDEs - Atom



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



# JavaScript IDEs - Atom



The screenshot shows the Atom IDE interface with the following details:

- Project Structure:** The left sidebar displays the project structure under "TestFlow". It includes sub-folders like ".idea", "AmdLoader", "c", "Flow", "UIKIT", "TestFlowWidget", "TodoDS", and "UIKIT". Inside "UIKIT" are "Input", "Button.js", and "ButtonModule.js". Inside "TestFlowWidget" are "Button.js", "Button.ts", "index.html", "test.js", and "tsconfig.json". Inside "TodoDS" are "UI" (containing ".TodoDS.index.mk") and "TodoDS.js". A file named ".flowconfig" is also listed.
- File Content:** The main editor tab is "index.html - TodoDS". The code is as follows:

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

At the bottom of the Atom interface, there are status indicators: "File 0 Project 0" (green checkmark), "No Issues" (green checkmark), "TodoDS/index.html 1:1" (green checkmark), "1 deprecation" (yellow warning icon), "UTF-8" (blue icon), "HTML" (blue icon), and "1 update" (blue icon).

# EditorConfig



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

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

# top-most EditorConfig file
root = true

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

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



**formation.tech**

# JavaScript (ECMAScript 3)

# JavaScript - Introduction



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

# JavaScript - ECMAScript



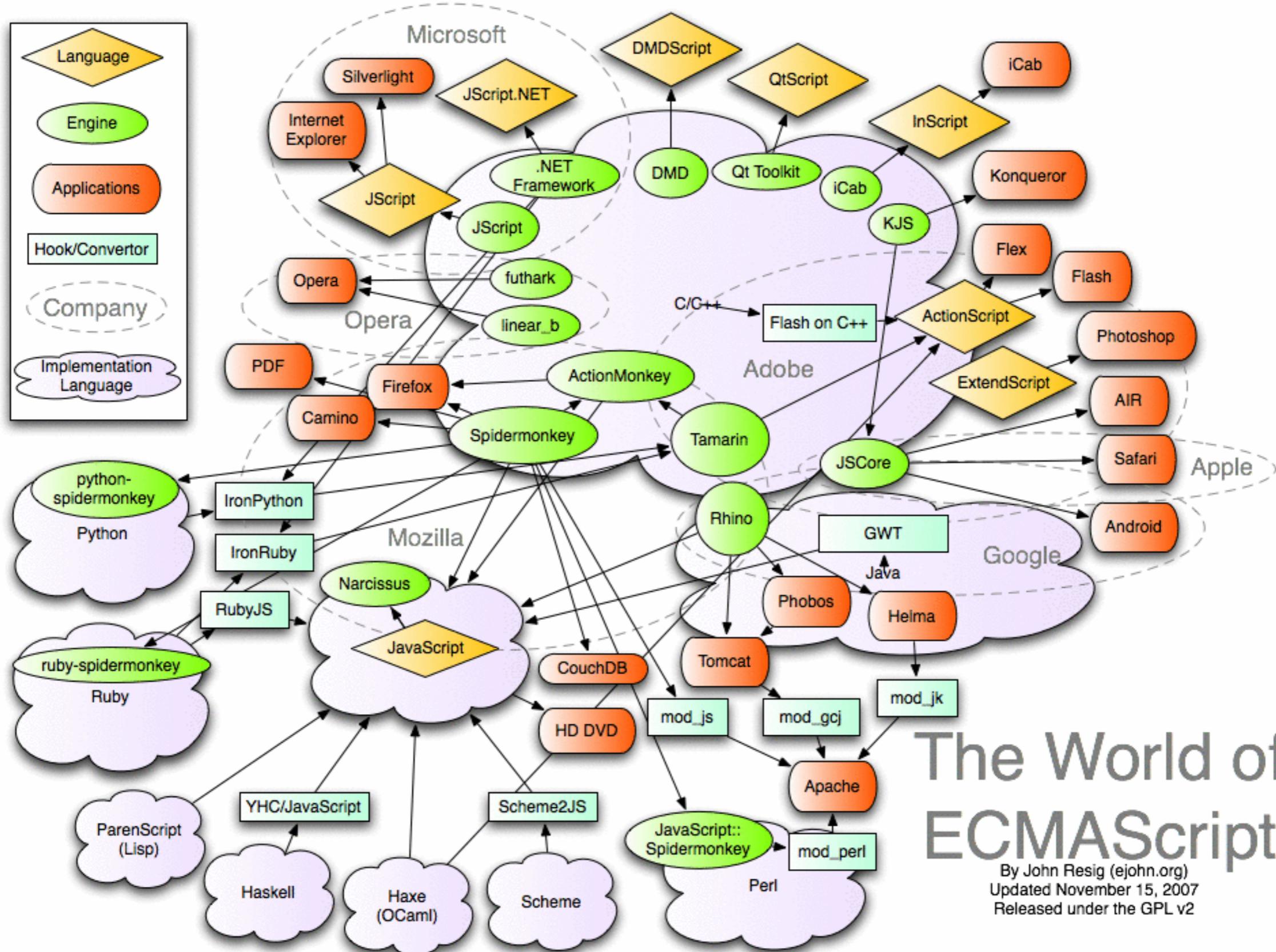
- JavaScript est une implémentation de la norme ECMAScript 262
- La norme la plus récente est ECMAScript 2019  
Aussi appelée ECMAScript 10 ou ES10 (juin 2019)  
<https://www.ecma-international.org/ecma-262/10.0/>
- Le langage a très fortement évolué avec ECMAScript 2015  
ECMAScript 6 / ES6 (juin 2015)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

# JavaScript - Documentation



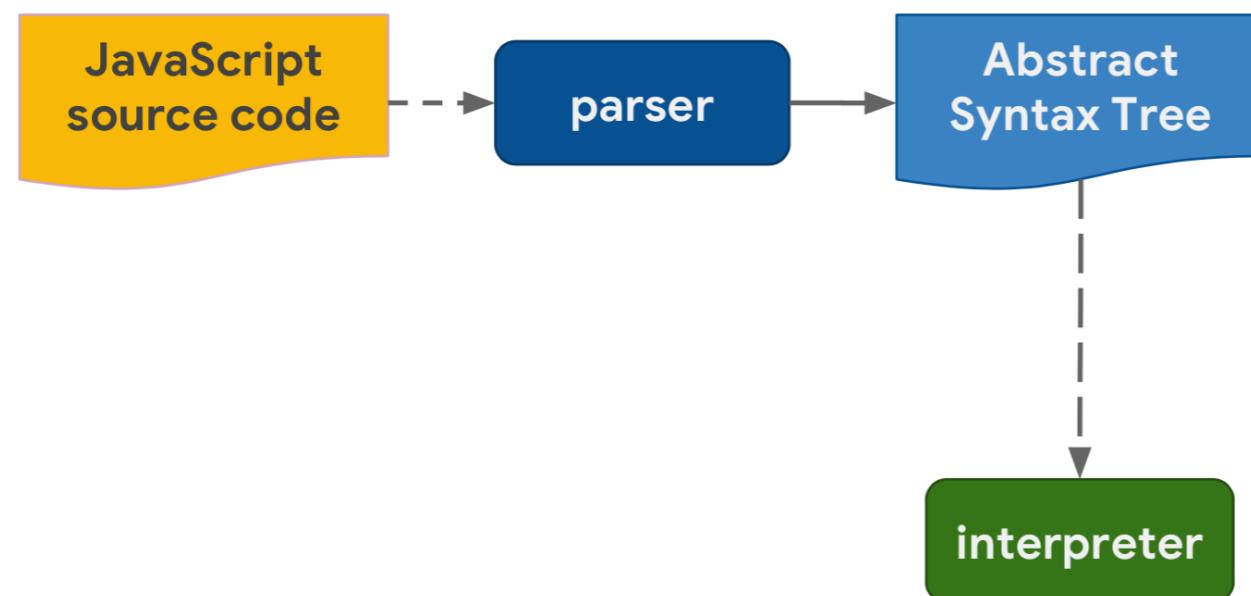
- La norme manque d'exemples et d'information sur les implémentations :  
<https://www.ecma-international.org/ecma-262/10.0/>
- Mozilla fournit une documentation open-source sur le langage JavaScript et sur les APIs Web (utiliser la version anglaise qui est plus à jour) :  
<https://developer.mozilla.org/en-US/docs/Web/JavaScript>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

# JavaScript - ECMAScript

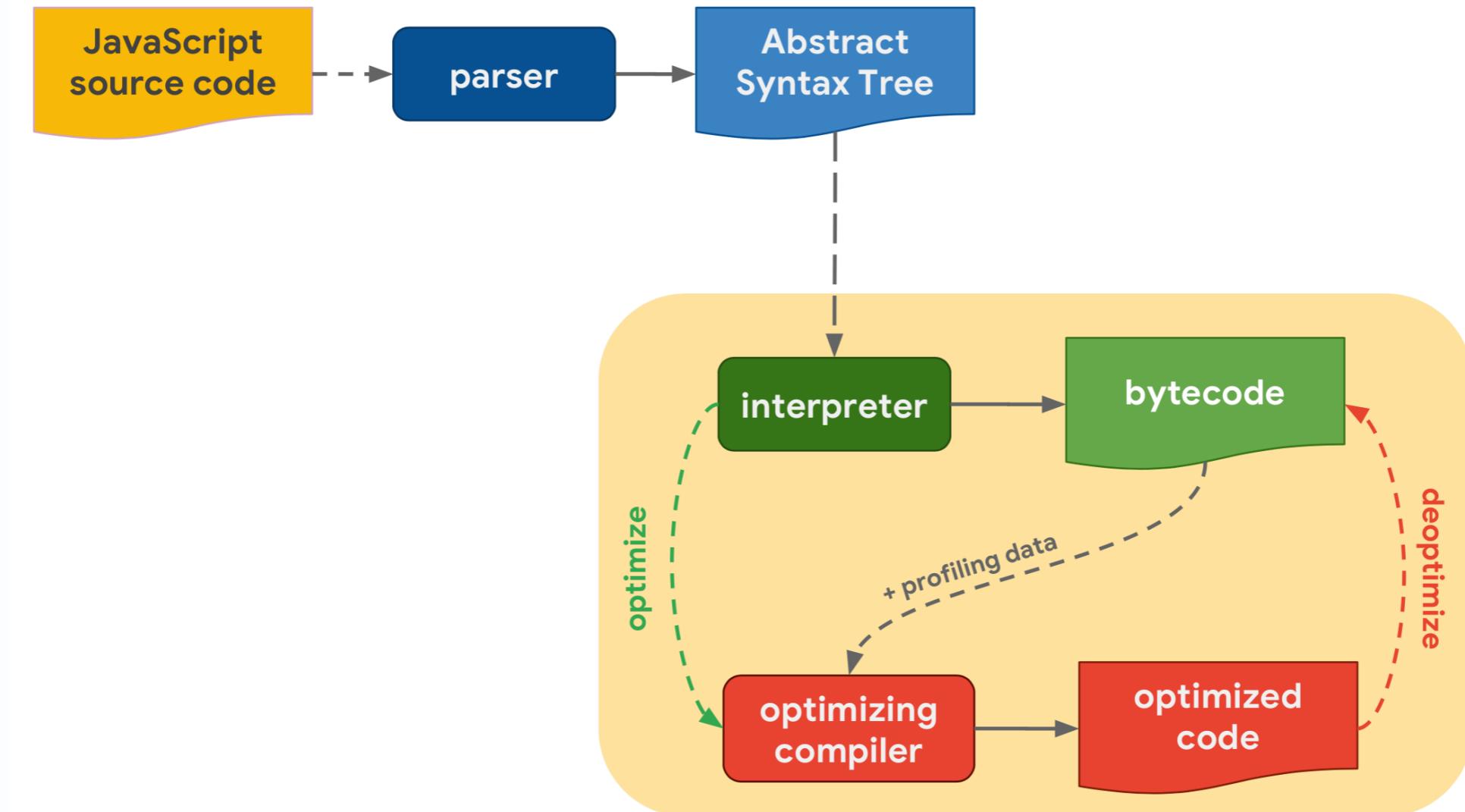




# JavaScript - Interprétation



# JavaScript - Compilation JIT



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

# JavaScript - Syntaxe



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

# JavaScript - Identifiants



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

# JavaScript - Mots clés



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

# JavaScript - Types



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

# JavaScript - Types



- Différence primitifs / complexes

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

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

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

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

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

# JavaScript - Number



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

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

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

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

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

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

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

# JavaScript - NaN



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

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

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

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

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

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

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

# JavaScript - Infinity



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

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

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

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

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

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

# JavaScript - Déclaration de variable



- Mot clé var

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

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

- Déclaration sans var

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

- ECMAScript 6

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

# JavaScript - Undefined



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

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

# JavaScript - Null



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

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

# JavaScript - Opérateurs



- Affectation

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

# JavaScript - Opérateurs



## ‣ Comparaison

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

# JavaScript - Opérateurs



## ▪ Arithmétiques

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

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

# JavaScript - Opérateurs



## ► Logiques

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

# JavaScript - Opérateurs



- Concaténation

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

- Ternaire

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

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

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

# JavaScript - Opérateurs



## ▸ Priorités

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



# JavaScript - Conversions

- Conversions implicites

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

- Conversions explicites

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

- Les conversions de types

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

# JavaScript - API



- Standard Built-in Objects

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

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

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

# JavaScript - Tableaux



- Structure et API

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

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

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

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

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

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

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

# JavaScript - Structures de contrôle



- if ... else

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

- switch

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

# JavaScript - Structures de contrôle



- while

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

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

- do ... while

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

console.log(alea);
```

- for

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

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



**formation.tech**

# Fonctions en JavaScript (ECMAScript 3)

# Fonctions en JavaScript - Introduction



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



# Fonctions en JavaScript - Syntaxe

- Function declaration

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

- Anonymous function expression

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

- Named function expression

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

# Fonctions en JavaScript - Function Declaration



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

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

# Fonctions en JavaScript - Function Expression



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

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

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

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

# Fonctions en JavaScript - Constantes



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

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

console.log(hello());

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

# Fonctions en JavaScript - Named Function Expression



- Anonymous function expression vs Named function expression

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

The screenshot shows a browser developer tools debugger interface. The code in the left pane is:

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

The line `12 console.log(firstName);` is highlighted with a blue selection bar. The right pane shows the call stack and scope:

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

Message: Paused on a JavaScript breakpoint.

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

The screenshot shows a browser developer tools debugger interface. The code in the left pane is:

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

The line `12 console.log(firstName);` is highlighted with a blue selection bar. The right pane shows the call stack and scope:

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

Message: Paused on a JavaScript breakpoint.

# Fonctions en JavaScript - Paramètres



- Paramètres

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

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

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

# Fonctions en JavaScript - Exceptions



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

```
var sum = function(a, b) {
  if (typeof a !== 'number' || typeof b !== 'number') {
    throw new Error('sum needs 2 number')
  }
  return a + b;
};

try {
  sum('1', '2'); // sum needs 2 number
}
catch (err) {
  console.log(err.message);
}
```

# Fonctions en JavaScript - Valeur par défaut



- Valeur par défaut

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

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

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

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

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

# Fonctions en JavaScript - Paramètres non déclarés



- Fonction Variadique

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

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

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

    return result;
};

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



# Fonctions en JavaScript - Imbrication

- Fonctions imbriquées

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

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



# Fonctions en JavaScript - Portées

- Portées

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

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

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



# Fonctions en JavaScript - Closure

- Closure

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

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

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

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

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

The line `3 console.log(msg);` is highlighted with a blue arrow and a blue background, indicating it is the current line of execution. The debugger sidebar on the right shows the call stack and scope information:

- Call Stack:
  - (anonymous function) closure.js:3
  - (anonymous function) closure.js:8
- Scope:
  - Local
    - this: Window
  - Closure (logClosure)
    - msg: "Hello"
  - Global
    - Infinity: Infinity
    - AnalyserNode: function AnalyserNode()
    - AnimationEvent: function AnimationEvent()

The message "Paused on a JavaScript breakpoint." is displayed at the bottom of the sidebar.



# Fonctions en JavaScript - Exemple de Closure

- › Sans Closure

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

- › Avec Closure

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

# Fonctions en JavaScript - Callbacks



- › Callback  
Lorsqu'un fonction est passée en paramètre d'entrée d'une autre fonction en vue d'être appelée plus tard, on parle de callback.
- › Callback synchrone / asynchrone  
Une fonction recevant un callback peut être synchrone, c'est à dire qu'elle doit s'exécuter entièrement avant d'appeler les instructions suivantes, ou asynchrone ce qui signifie que la fonction sera appelée dans un prochain passage de la « boucle d'événements »

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

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

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

# Fonctions en JavaScript - Callback Synchrone



- API recevant un callback synchrone

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

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

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

console.log('After forEachSync');

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

# Fonctions en JavaScript - Callback Asynchrone



- API recevant un callback asynchrone

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

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

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

console.log('After forEachASync');

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

# Fonctions en JavaScript - Boucle d'événements



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

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

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

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

# Fonctions en JavaScript - Boucle d'événements



- Boucle d'événements

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

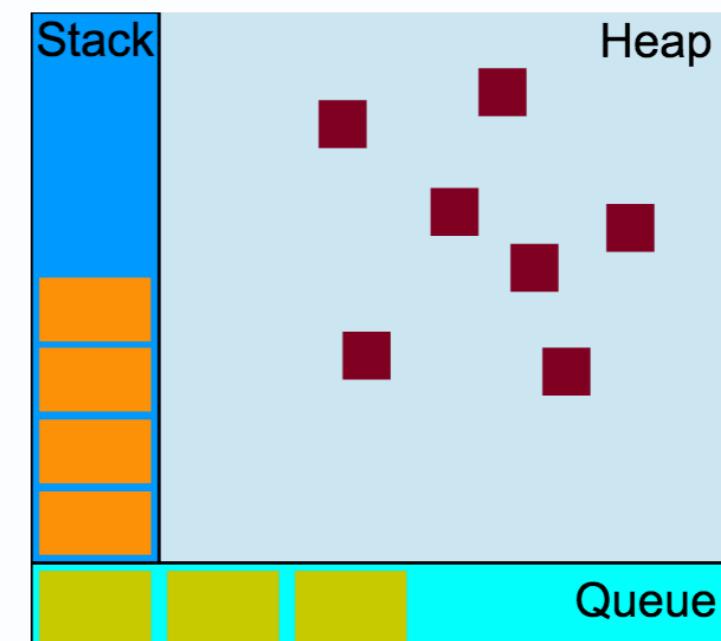
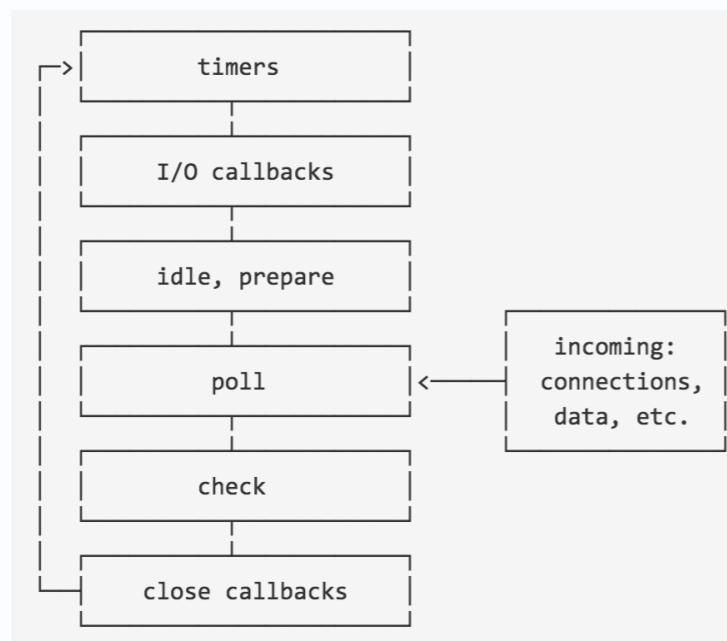
Tant qu'il y a des appels en cours dans la pile d'appels, où des callbacks en attente dans la file de callback, on ne passe pas à la prochain itération. Dans le navigateur, un seul thread est en charge du JavaScript et du rendu, pour un rendu à 60FPS il faut qu'un passage dans la boucle JS + rendu ne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

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

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

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

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

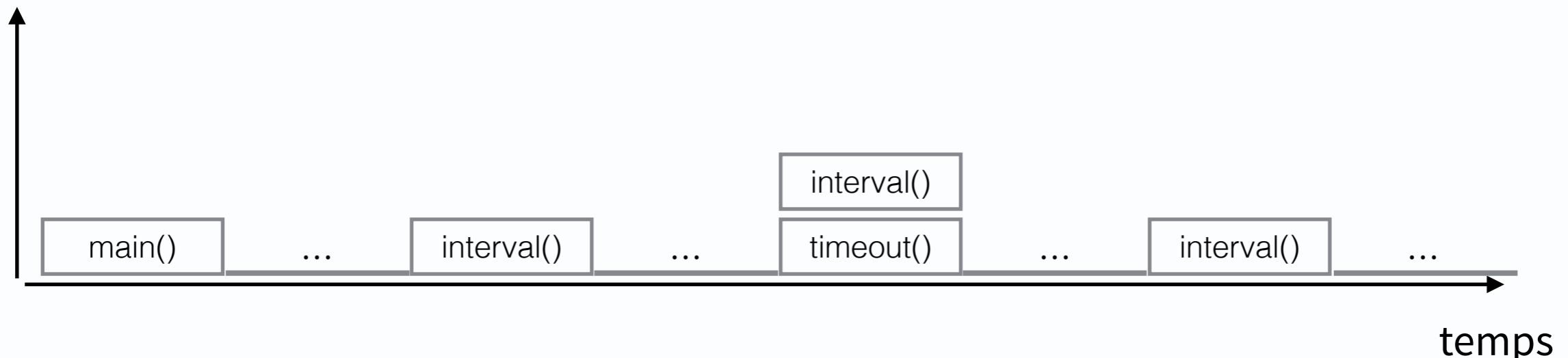




# Fonctions en JavaScript - Boucle d'événements

- Boucle d'événements

File d'attente



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

# Fonctions en JavaScript - API Function



- Object function

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

# Fonctions en JavaScript - Modules



- ▶ 

## Module

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

- ▶ 

## Immediately Invoked Function Expression (IIFE)

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

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

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

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

# Fonctions en JavaScript - Exercice



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



**formation.tech**

# JavaScript Orienté Objet (ECMAScript 3)



# JavaScript Orienté Objet - Introduction

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



# JavaScript Orienté Objet - Objets préinstanciés

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

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

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

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

# JavaScript Orienté Objet - Extensibilité



- Extensibilité

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

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

- On peut également modifier ou supprimer des propriétés

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

# JavaScript Orienté Objet - Objets ponctuels



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

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

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

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

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

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



# JavaScript Orienté Objet - Opérateurs

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

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

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

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

# JavaScript Orienté Objet - Fonction Constructeur



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

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

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

var instructor = new Person('Romain');

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

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

# JavaScript Orienté Objet - Fonction Constructeur



- En utilisant une fonction constructeur + son prototype :

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

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

var instructor = new Person('Romain');

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

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

# JavaScript Orienté Objet - Héritage



- En utilisant une fonction constructeur + son prototype :

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

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

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

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

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

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

# JavaScript Orienté Objet - Prototype



- Définition Wikipedia :  
La programmation orientée prototype est une forme de programmation orientée objet sans classe, basée sur la notion de prototype. Un prototype est un objet à partir duquel on crée de nouveaux objets.
- Comparaison des modèles à classes et à prototypes
  - Objets à classes :
    - Une classe définie par son code source est statique ;
    - Elle représente une définition abstraite de l'objet ;
    - Tout objet est instance d'une classe ;
    - L'héritage se situe au niveau des classes.
  - Objets à prototypes :
    - Un prototype défini par son code source est mutable ;
    - Il est lui-même un objet au même titre que les autres ;
    - Il a donc une existence physique en mémoire ;
    - Il peut être modifié, appelé ;
    - Il est obligatoirement nommé ;
    - Un prototype peut être vu comme un exemplaire modèle d'une famille d'objet ;
    - Un objet hérite des propriétés (valeurs et méthodes) de son prototype ;

# JavaScript Orienté Objet - Prototype



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

# JavaScript Orienté Objet - JSON



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

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



# JavaScript Orienté Objet - JSON vs XML

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

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

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

# JavaScript Orienté Objet - JSON



- JavaScript depuis ECMAScript 5 fourni l'objet global JSON qui contient 2 méthodes, parse (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 = {  
    ...  
};  
  
function Jeu(options) { ... };  
  
// ....  
  
const game = new Jeu({  
    min: 0,  
    max: 100  
});  
  
game.jouer();
```



**formation.tech**

# Expression Régulières (ECMAScript 3)

# Expression Régulières - Introduction



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

# Expression Régulières - Création



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

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

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



# Expression Régulières - Format

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

# Expression Régulières - Format



- On peut également indiquer une liste de caractères [a-f] est l'équivalent [abcdef] (les caractères doivent se suivrent dans le code ASCII) :
  - / [a-z]\$ / va matcher **a**rte mais pas tf1
  - / ^ [a-fw-z] / va matcher **a**vion, **w**agon mais pas piéton
  - / ^ [0-9] / va matcher **2**019 mais pas bateau
- Les expressions régulières sont sensibles à la casse, il faudra donc inclure la majuscule et la minuscule dans le méta-caractère :
  - / ^ [Rr] / va matcher **R**omain et **r**oman
- Les caractères accentués ne sont pas inclus dans la liste [a-z], il faudra utiliser leur code Unicode (<https://unicode-table.com/fr/>) :
  - / ^ [\u00c0-\u00ff] / va matcher **\u00e7**a ou **\u00e0**

# Expression Régulières - Format



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



# Expression Régulières - Format

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



# Expression Régulières - Format

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



# Expression Régulières - Format

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



# Expression Régulières - regex.test

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

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

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

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

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



# Expression Régulières - regex.exec

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

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

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

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



# Expression Régulières - string.match

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

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



# Expression Régulières - string.split

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

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



# Expression Régulières - string.search

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

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



# Expression Régulières - string.replace

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

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

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



**formation.tech**

# ECMAScript 5.1

# ECMAScript 5.1 - Introduction



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



# ECMAScript 5.1 - Mode Strict

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

# ECMAScript 5.1 - Mode Strict



- Activer le mode strict

- Globalement

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

- A partir d'une ligne

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

- Dans une fonction

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

# ECMAScript 5.1 - Mode Strict



- Mots clés réservés

- Sloppy Mode

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

- Strict Mode

```
'use strict';

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

# ECMAScript 5.1 - Mode Strict



- › Oubli du mot clé var

- Sloppy Mode

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

- Strict Mode

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

# ECMAScript 5.1 - Mode Strict



- › Désactivation de with

- Sloppy Mode

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

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

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

- Strict Mode

```
'use strict';

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

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

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

# ECMAScript 5.1 - Mode Strict



- Pas d'identifiant dans eval

- Sloppy Mode

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

- Strict Mode

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

# ECMAScript 5.1 - Mode Strict



- Supprimer des variables

- Sloppy Mode

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

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

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

- Strict Mode

```
'use strict';

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

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

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

# ECMAScript 5.1 - Mode Strict



- Utilisation de this

- Sloppy Mode

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

var contact = Contact('Romain');

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

- Strict Mode

```
'use strict';

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

var contact = Contact('Romain');

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

# ECMAScript 5.1 - Immutable globals



- Nouvelles variables globales non modifiables

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

# ECMAScript 5.1 - Array



- › Programmation fonctionnelle

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

- › Tableaux

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

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

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

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

# ECMAScript 5.1 - Function.prototype.bind



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

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

# ECMAScript 5.1 - JSON



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

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



# ECMAScript 5.1 - Trailing commas

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

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

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



# ECMAScript 5.1 - get syntax

- On peut masquer une méthode derrière une propriété en lecture

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

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

# ECMAScript 5.1 - set syntax



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

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

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

# ECMAScript 5.1 - Object.getPrototypeOf



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

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

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

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



# ECMAScript 5.1 - Object.defineProperty

- Permet une définition plus fine d'une propriété

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

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

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

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

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

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



# ECMAScript 5.1 - Object.defineProperty

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

```
'use strict';

var contact = {
  firstName: 'Romain'
};

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

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



# ECMAScript 5.1 - Object.defineProperty

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

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

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

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

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

# ECMAScript 5.1 - Object.keys



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

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



# ECMAScript 5.1 - Object.preventExtensions

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

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

# ECMAScript 5.1 - Object.preventExtensions



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

```
'use strict';

var contact = {
  firstName: 'Romain'
};

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

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



# ECMAScript 5.1 - Verrous

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

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

# ECMAScript 5.1 - Héritage en ES5



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

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

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

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

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

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



**formation.tech**

# ECMAScript 6 / ECMAScript 2015

# ECMAScript 6 - Introduction



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

# ECMAScript 6 - Compatibilité



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

# ECMAScript 6 - Portées de bloc



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

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

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

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

# ECMAScript 6 - Portées de bloc



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

```
'use strict';

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

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

# ECMAScript 6 - Constantes



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

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

# ECMAScript 6 - Template literal



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

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

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

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



# ECMAScript 6 - new.target

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

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

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

# ECMAScript 6 - Fonctions fléchées



## ‣ Arrow Functions

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

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

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

# ECMAScript 6 - Fonctions fléchées



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

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

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

  return 'Won !';
};
```

# ECMAScript 6 - Fonctions fléchées



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

```
const globalThis = this;

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

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

# ECMAScript 6 - Default Params



- Paramètres par défaut

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

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

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

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

# ECMAScript 6 - Default Params



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

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

# ECMAScript 6 - Rest Parameters



## ▶ Paramètres restants

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

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

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

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

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

# ECMAScript 6 - Spread Operator



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

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

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

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

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

# ECMAScript 6 - Shorthand property



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

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

const coords = {
  x,
  y,
};

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

# ECMAScript 6 - Method properties



- Method properties
  - Syntaxe simplifiée pour déclarer des méthodes

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

# ECMAScript 6 - Computed Property Names



- Computed Property Names

Permet d'utiliser une expression en nom de propriété

```
let i = 0;

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

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

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

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

# ECMAScript 6 - Array Destructuring



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

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

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

# ECMAScript 6 - Array Destructuring



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

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

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

# ECMAScript 6 - Object Destructuring



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

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

# ECMAScript 6 - Object Destructuring



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

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

# ECMAScript 6 - Mot clé class



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

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

```
// ES5  
// var Person = function(firstName) {  
//     this.firstName = firstName;  
// };  
// Person.prototype.hello = function() {  
//     return 'Hello my name is ' + this.firstName;  
// };
```

# ECMAScript 6 - Mot clé class



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

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



# ECMAScript 6 - Boucle for .. of

- **Boucle for .. of**
  - Permet de boucler sur des objets itérables (Array, Map, Set, String, TypedArray, arguments)

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

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

# ECMAScript 6 - Object.assign



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

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

# ECMAScript 6 - Object.assign



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

```
const deepClone = function(obj) {  
  let clone = Object.assign({}, obj);  
  
  for (let p in obj) {  
    if (obj.hasOwnProperty(p) && typeof obj[p] === 'object') {  
      clone[p] = deepClone(obj[p]);  
    }  
  }  
  
  return clone;  
};  
  
const deepCopy = deepClone(obj);  
console.log(deepCopy.address === obj.address); // false
```

# ECMAScript 6 - Générateurs



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

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

# ECMAScript 6 - Symbol



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

```
var locale = {
  fr_FR: Symbol(),
  en_US: Symbol()
};

var translations = {
  [locale.fr_FR]: {
    'hello': 'bonjour',
    'cat': 'chat'
  },
  [locale.en_US]: {
    'hello': 'hello',
    'cat': 'cat'
  }
};

var translate = function (key, locale = locales.en_US) {
  return translations[locale][key];
};

console.log(translate('hello', locale.fr_FR)); // bonjour
```

# ECMAScript 6 - Symbol



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

```
class Collection {  
    constructor() {  
        this.list = [];  
    }  
    add(elt) {  
        this.list.push(elt);  
        return this;  
    }  
    *[Symbol.iterator]() {  
        for (let elt of this.list) {  
            yield elt;  
        }  
    }  
}  
  
let firstNames = new Collection();  
firstNames.add('Romain').add('Eric');  
  
for (let firstName of firstNames) {  
    console.log(firstName); // Romain Eric  
}
```

# ECMAScript 6 - Exercice



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



**formation.tech**

# JavaScript Asynchrone

# JavaScript Asynchrone - Introduction



- Boucle d'événement

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

- Avantages

- Gestion de la concurrence simplifiée
- Performance

- Inconvénients

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

# JavaScript Asynchrone - Perte de contexte



- › Où est this ?

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

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

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

# JavaScript Asynchrone - Perte de contexte



- Dans l'implémentation setTimeout de Node.js :

<https://github.com/nodejs/node/blob/v6.x/lib/timers.js#L382>

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



# JavaScript Asynchrone - Perte de contexte

- › Solution 1 : Sauvegarder this dans la portée de closure  
La valeur de this peut être sauvegardée dans la portée de closure, la variable s'appelle généralement that (ou \_this, self, me...)

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

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



# JavaScript Asynchrone - Perte de contexte

- Solution 2 : Function.bind (ES5)

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

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

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

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

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

# JavaScript Asynchrone - Perte de contexte



- › Solution 3 : Arrow Function (ES6)

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

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

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



# JavaScript Asynchrone - Callback Hell

- Callback Hell / Pyramid of doom

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

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

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

# JavaScript Asynchrone - Callback Hell



- Callback Hell

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

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

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

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

# JavaScript Asynchrone - Async



- **Async**

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

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

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

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

# JavaScript Asynchrone - Promesses



- Exemple avancé

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

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

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

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

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

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

# JavaScript Asynchrone - Promesses



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

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

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

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

# JavaScript Asynchrone - Observables



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

# JavaScript Asynchrone - Observables



- Exemple Promise vs Observable

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

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

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

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

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



# JavaScript Asynchrone - Exercice

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



**formation.tech**

# ECMAScript 7 / ECMAScript 2016

# ECMAScript 7 - Introduction



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

# ECMAScript 7 - Opérateur d'exponentiation



## ‣ Opérateur d'exponentiation

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

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

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

## ‣ Plugin babel : transform-exponentiation-operator

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

# ECMAScript 7 - Array.prototype.includes



## ‣ Array.prototype.includes

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

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

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

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

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



**formation.tech**

# ECMAScript 8 / ECMAScript 2017

# ECMAScript 8 - Introduction



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

# ECMAScript 8 - Virgules finales sur les fonctions



- Virgules finales sur les fonctions

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

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

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



# ECMAScript 8 - Virgules finales sur les fonctions

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

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

- Object literal

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

# ECMAScript 8 - Virgules finales sur les fonctions



- Exemple de diff sans et avec virgule finale

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

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

# ECMAScript 8 - Fonctions async / await



- **async / await**

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

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

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

# ECMAScript 8 - Fonctions async / await



- › Sans async / await

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

- › Avec async / await

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



# ECMAScript 8 - Object.values / Object.entries

- **Object.values**

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

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

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

- **Object.entries**

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

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

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

# ECMAScript 8 - String Padding



- String Padding

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

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

# ECMAScript 8 - Atomics



- Méthodes statiques de l'objet `Atomics`

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

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

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

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

# ECMAScript 8 - Object.getOwnPropertyDescriptors



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

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

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

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



**formation.tech**

# ECMAScript 9 / ECMAScript 2018

# ECMAScript 9 - Rest/Spread Properties



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

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

# ECMAScript 9 - Template Literal Revision



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

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

# ECMAScript 9 - Regexp DotAll Flag



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

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

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

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

# ECMAScript 9 - Regexp Capture Group



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

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

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

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

# ECMAScript 9 - Regexp Lookbehind



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

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

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

# ECMAScript 9 - Regexp Unicode property escapes



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

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

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

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

# ECMAScript 9 - Promise.prototype.finally



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

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

let filehandle;

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

# ECMAScript 9 - Promise.prototype.finally



- Avec les fonctions asynchrones

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

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

# ECMAScript 9 - Asynchronous iteration



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

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

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

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

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

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

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



**formation.tech**

# ECMAScript 10 / ECMAScript 2019

# ECMAScript 10 - Optional Catch Binding



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

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

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

# ECMAScript 10 - JSON Superset



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

# ECMAScript 10 - Symbol.prototype.description



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

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

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



# ECMAScript 10 - Object.fromEntries

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

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



# ECMAScript 10 - JSON.stringify

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

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

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

# ECMAScript 10 - trimStart / trimEnd



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

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

# ECMAScript 10 - flat / flatMap



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

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

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

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

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

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



**formation.tech**

# ECMAScript 11 / ECMAScript 2020

# ECMAScript 11 - String.prototype.matchAll



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

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

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

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

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



# ECMAScript 11 - import()

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

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

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

# ECMAScript 11 - Promise.allSettled



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

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

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

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



**formation.tech**

# ECMAScript Next / ESNext

# ECMAScript Next - Introduction



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





# ECMAScript Next - globalThis

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

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

# ECMAScript Next - BigInt



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

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

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

# ECMAScript Next - Class Properties



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

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

- Les propriétés préfixées par un dièse seraient privées

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

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

# ECMAScript Next - Optional Chaining



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

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



**formation.tech**

# DOM

# DOM - Introduction

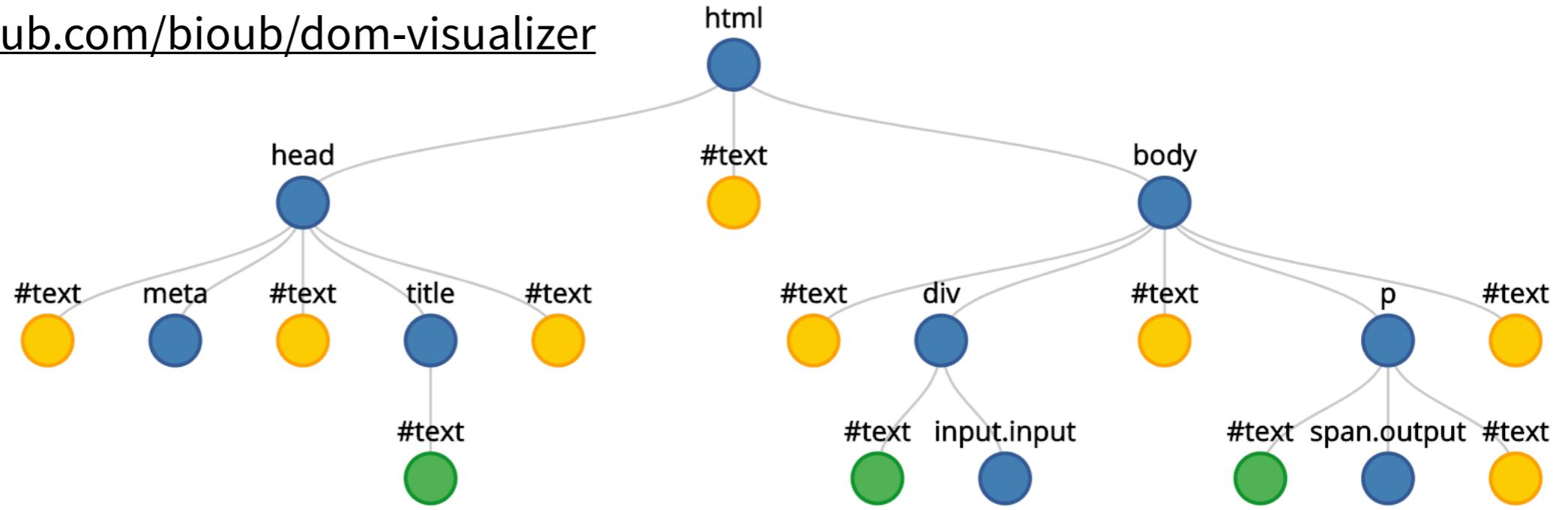


- Document Object Model  
API créé par Netscape en même temps que le JavaScript qui permet la manipulation du HTML en mémoire sous la forme d'un arbre
- W3C  
Une norme existe depuis 1998, aujourd'hui dans sa 4e édition  
DOM Level 4 : <http://www.w3.org/TR/dom/>  
DOM Level 3 Events : <http://www.w3.org/TR/DOM-Level-3-Events/>  
HTML5 : <http://www.w3.org/TR/html5/>



# DOM - Arbre

- Représentation sous la forme d'un arbre  
<https://github.com/bioub/dom-visualizer>



```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>DOM</title>
</head>
<body>
  <div>Name : <input class="input"></div>
  <p>Hello <span class="output"></span> !</div>
</body>
</html>
```

# DOM - Level 1



- Helloworld en DOM 1 (années 1990)

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>DOM</title>
  <script>
    function helloworld() {
      var inputElt = document.getElementsByTagName('input')[0];
      var spanElt = document.getElementsByTagName('span')[0];
      var text = document.createTextNode(inputElt.value);
      if (!spanElt.childNodes.length) {
        spanElt.appendChild(text);
      } else {
        spanElt.replaceChild(text, spanElt.firstChild);
      }
    }
  </script>
</head>
<body>
  <div>Name : <input id="input" onkeyup="helloworld()"></div>
  <p>Hello <span id="output"></span> !</p>
</body>
</html>
```



- Helloworld en DOM 1 (années 1990)
- Inconvénients :
  - écouter un événement sous forme d'attribut nécessite que la fonction helloworld soit globale
  - méthodes limités pour retrouver un élément dans l'arbre
  - méthodes limités pour créer du contenu (lourdeur de créer manuellement le noeud de texte)

# DOM - Level 2 et 3



- Helloworld en DOM 2 - 3 (années 2000)

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>DOM</title>
    <script>
        window.onload = function() {
            var inputElt = document.getElementById('input');
            var spanElt = document.getElementById('output');

            inputElt.onkeyup = function helloworld() {
                spanElt.textContent = this.value;
            };
        };
    </script>
</head>
<body>
    <div>Name : <input id="input"></div>
    <p>Hello <span id="output"></span> !</div>
</body>
</html>
```



- Helloworld en DOM 2 - 3 (années 2000)
- Inconvénients :
  - le code est en général en début de fichier, et s'exécute au chargement, il faut donc attendre que la page ait fini de charger
  - la méthode getElementById n'est disponible que sur l'objet document
  - la propriété onkeyup est unique, on risque un conflit si plusieurs callbacks écoutent le même événement sur le même élément (peut provenir d'une extension du navigateur)
  - on ne peut pas écouter l'événement dans une Event Phase spécifique

# DOM - Level 4



- Helloworld en DOM 4 (années 2010)

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>DOM</title>
</head>
<body>
  <div>Name : <input id="input"></div>
  <p>Hello <span id="output"></span> !</div>
  <script>
    (function() {
      'use strict';
      var inputEl = document.querySelector('#input');
      var spanEl = document.querySelector('#output');

      inputEl.addEventListener('input', function helloworld() {
        spanEl.innerText = this.value;
      });
    }());
  </script>
</body>
</html>
```



- Helloworld en DOM 4 (années 2010)
- Inconvénients :
  - module IIFE pour éviter les variables globales
  - mode strict pour désactiver des mauvais comportement
  - ambiguïté du mot clé this, l'objet dans lequel on est ? le champ input ?

# DOM - Level 4 + ES2015



- Helloworld en DOM 4 + ES2015 (années 2015+)

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>DOM</title>
</head>
<body>
  <div>Name : <input id="input"></div>
  <p>Hello <span id="output"></span> !</div>
  <script type="module">
    const inputEl = document.querySelector('#input');
    const spanEl = document.querySelector('#output');

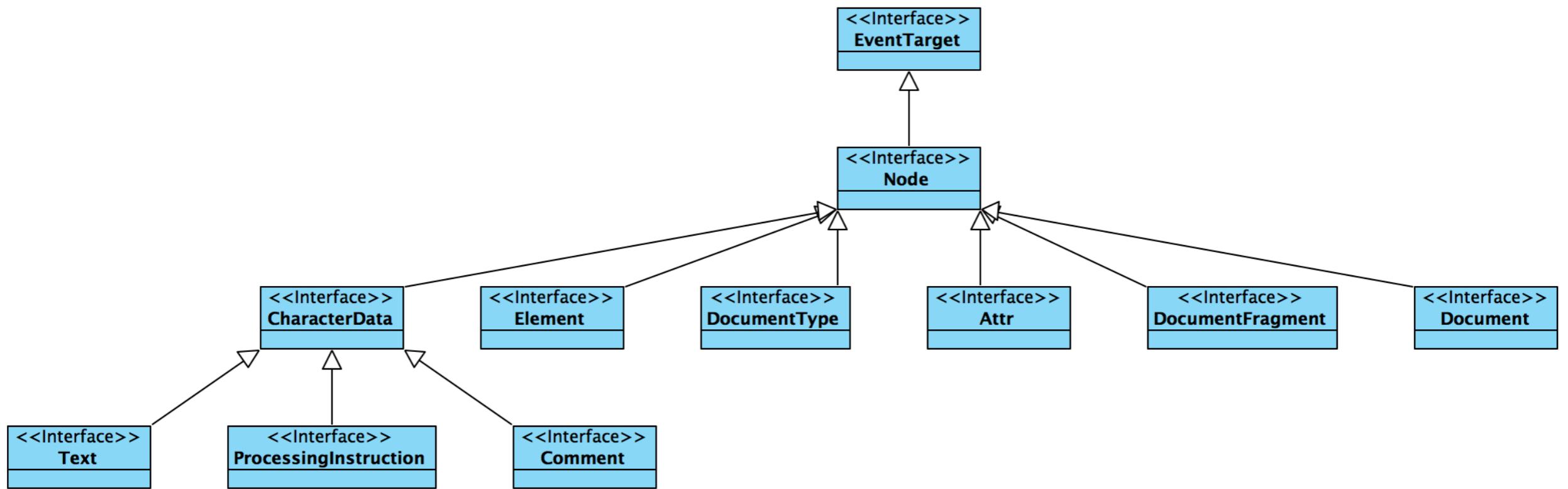
    inputEl.addEventListener('input', (event) => {
      spanEl.innerText = event.target.value;
    });
  </script>
</body>
</html>
```

# DOM - Level 4

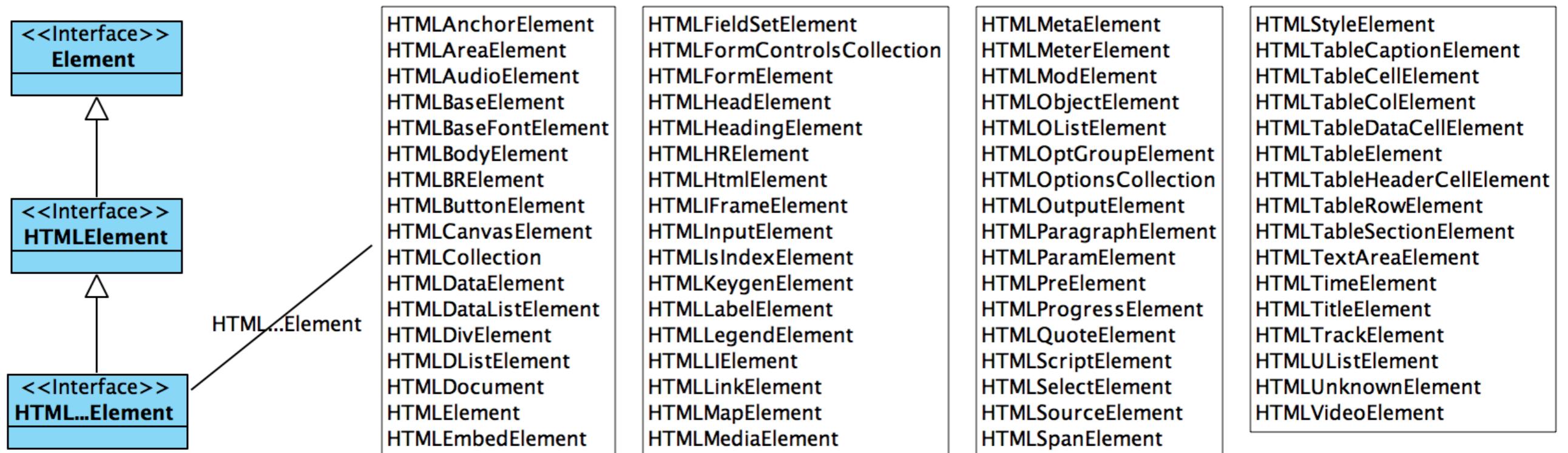


- Helloworld en DOM 4 (années 2010)
- Inconvénients :
  - les nouvelles syntaxe perdent la compatibilité avec les navigateurs ancien
  - type="module" également

# DOM - Interfaces



# DOM - HTML Interfaces





# DOM - Parcourir l'arbre

- Depuis n'importe quel noeud Element (DOM 4, IE9+) :
  - children : les noeuds Element enfants
  - firstElementChild : le premier noeud Element enfant
  - lastElementChild : le dernier noeud Element enfant
  - previousElementSibling : le frère Element précédent (même parent)
  - nextElementSibling : le frère Element suivant (même parent)
  - parentElement : le parent

## Browser compatibility

Desktop	Mobile	Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari
		Basic support (on <a href="#">Element</a> )	1.0	3.5 (1.9.1)	9.0	10.0	4.0
		Support on <a href="#">Document</a> and <a href="#">DocumentFragment</a> <small>⚠</small>	29.0	25.0 (25.0)	Not supported	16.0	Not supported

# DOM - Rechercher dans l'arbre



- Rechercher le premier noeud qui matche un sélecteur CSS
  - `document.querySelector('selecteur')`
  - `element.querySelector('selecteur')`
- Rechercher tous les éléments matchent un sélecteur CSS (retourne un NodeList, itérable)
  - `document.querySelectorAll('selecteur')`
  - `element.querySelectorAll('selecteur')`

# DOM - Raccourcis



- Tous les éléments : `document.all`
- Body : `document.body`
- Head : `document.head`
- Forms : `document.forms[ position ]` ou `document.forms[ id ]`
- Images : `document.images[ position ]` ou `document.images[ id ]`
- Scripts : `document.scripts[ position ]` ou `document.scripts[ id ]`
- Title : `document.title`



# DOM - Lire / écrire du contenu

- Entre la balise ouvrante / fermante
  - Element.textContent
  - Element.innerText
  - Element.innerHTML
- Valeur d'un champs (input, select, textarea)
  - Element.value

# DOM - Attributs



- Sauf exception : les propriétés d'un Element porte le nom de l'attribut
- Ex : Element.id, HTMLElement.lang, HTMLElement.title, HTMLFormElement.action, HTMLFormElement.name...
- Exceptions :
  - Element.className (class est un mot clé de JS),
  - HTMLMetaElement n'a pas de propriété charset
  - ...
- Pour accéder à n'importe quel attribut :
  - Element.getAttribute( name )
  - Element.setAttribute( name, value )



# DOM - Attributs booléens

- Certains attributs ont un caractère booléen
- En HTML5 :
  - <input type="text" required>
- En XHTML
  - <input type="text" required="required">
- DOM :
  - Element.required (true/false)

# DOM - Datasets



- Pour stocker une valeur dans une balise dans un attribut "custom"
- DOM
  - `Element.dataSet.monAttribut = "Valeur"`
- HTML
  - `<balise data-mon-attribut="valeur">`



# DOM - Ajouter des noeuds

- Pour créer un noeud on utilise une méthode commencer par *create* *document.createElement*, *document.createTextNode*, *document.createAttribute*...
- Pour l'insérer on utilise les méthodes *parent.insertBefore* ou *parent.appendChild*
- De nouvelles méthodes sont en train de faire leur apparition (inspirée de jQuery) : *append*, *prepend*, *after*, *before*, *remove*, *replace*

# DOM - Événements



- Les événements sont un mécanisme permettant d'exécuter du code au moment où une action se produit
- L'action peut être utilisateur (ex : clic) ou liée à un concept informatique (ex : la réponse d'une requête AJAX est reçue)
- Certains événements peuvent s'appliquer à tous les éléments (click, mousemove, keyup), d'autres sont spécifiques à certains éléments (submit d'un formulaire, timeupdate d'une vidéo...)
- Pour écouter des événements du DOM on utilise la méthode *addEventListener*:

```
inputElt.addEventListener('input', (event) => {
  spanElt.innerText = event.target.value;
});
```

# DOM - Événements

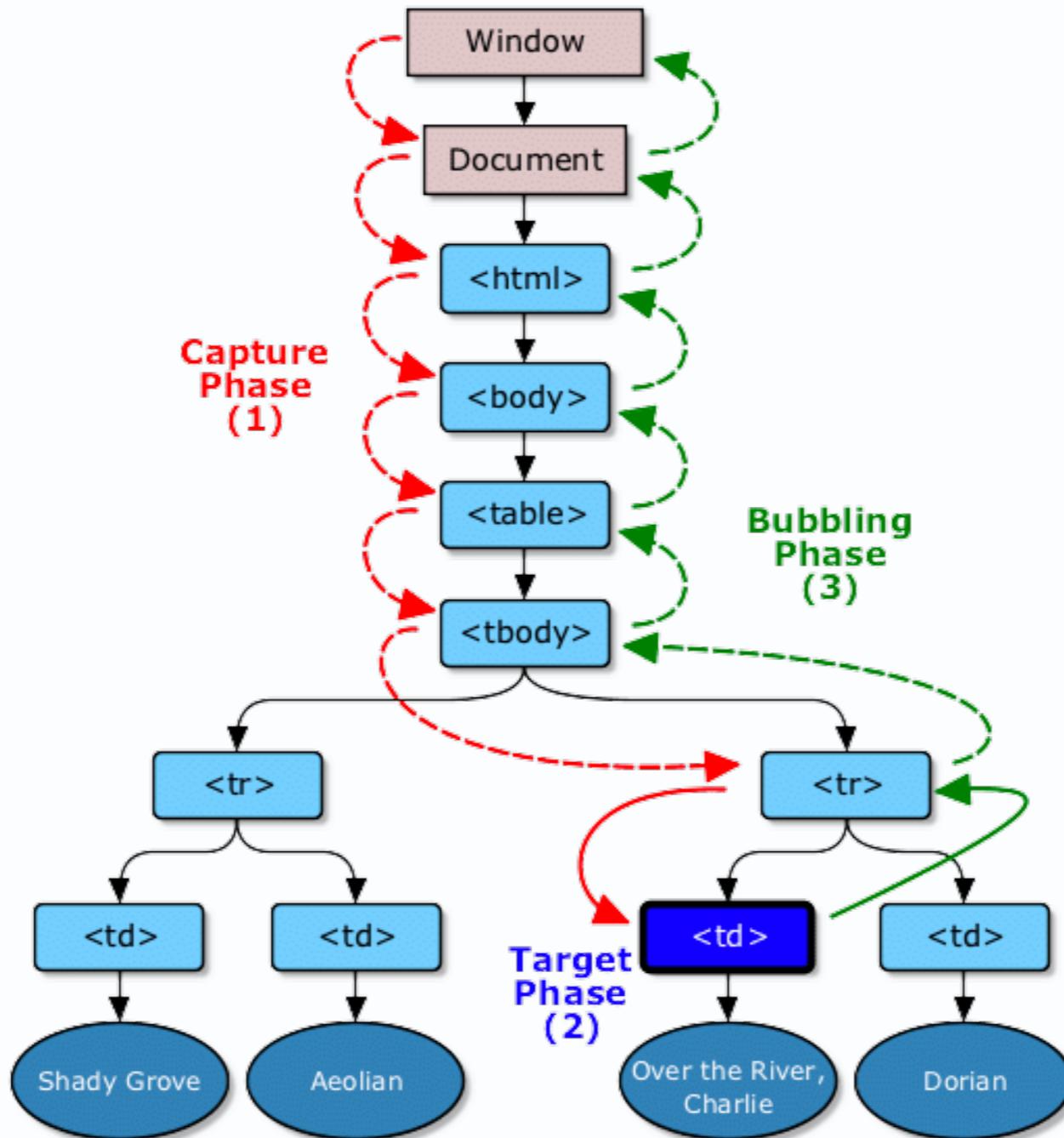


- Principaux événements souris :  
click, dblclick, mousedown, mousemove, mouseup, mouseenter, mouseleave,  
mouseover, mouseout
- Principaux événements clavier :  
keydown, keypress, keyup
- Principaux événements liés aux formulaires :  
submit, reset, focus, blur, input, change
- TouchEvents : événement tactiles (Smartphones, Tablettes...)
- PointerEvents : abstraction sur l'interface de pointeur (Souris, Tactile, Stylet...)

# DOM - Événements



- Pour les événements du DOM on peut distinguer différentes phases





# DOM - Événements

- › Pour écouter dans la phase de bubbling :

```
<body>
  <button class="btn">Clic moi</button>
  <div class="output"></div>
  <script type="module">
    const btnElt = document.querySelector('.btn');
    const outputElt = document.querySelector('.output');
    btnElt.addEventListener('click', () => {
      outputElt.textContent += 'button ';
    });
    document.body.addEventListener('click', () => {
      outputElt.textContent += 'body ';
    });
    // au clic du bouton : button body
  </script>
</body>
```



# DOM - Événements

- › Pour écouter dans la phase de capture :

```
<body>
  <button class="btn">Clic moi</button>
  <div class="output"></div>
  <script type="module">
    const btnElt = document.querySelector('.btn');
    const outputElt = document.querySelector('.output');
    btnElt.addEventListener('click', () => {
      outputElt.textContent += 'button ';
    });
    document.body.addEventListener('click', () => {
      outputElt.textContent += 'body ';
    }, true); // ou { useCapture: true }
    // au clic du bouton : body button
  </script>
</body>
```



# DOM - Événements

- › Pour écouter dans la phase de target :

```
<body>
  <button class="btn">Clic moi</button>
  <div class="output"></div>
  <script type="module">
    const outputElt = document.querySelector('.output');
    document.body.addEventListener('click', (event) => {
      if (event.target.classList.contains('btn')) {
        outputElt.textContent += 'button ';
      }
    });
    // au clic du bouton (uniquement) : button
  </script>
</body>
```

- › Moins lisible mais permet d'écouter le clic du bouton y compris s'il n'existe pas encore

# DOM - Événements



- Lorsqu'on écoute un événement du DOM, le callback est appelé avec un objet de type Event
- Il existent un certain nombre d'interface qui dérivent d'Event :  
<https://developer.mozilla.org/en-US/docs/Web/API/Event#Introduction>
- Exemple : MouseEvent, KeyboardEvent, PointerEvent...
- Un événement de type MouseEvent contiendra des propriétés telles que clientX, clientY (position de la souris sur la page)

# DOM - Événements



- La méthode preventDefault permet d'empêcher l'action par défaut du navigateur lorsqu'il y en a une
- Exemple : submit d'un formulaire, click sur un lien, début de sélection de texte (mousedown)...

```
Number : <input class="number"/>
<script type="module">
  const inputEl = document.querySelector('.number');
  inputEl.addEventListener('beforeinput', (event) => {
    if (!event.data.match(/\d/)) {
      event.preventDefault();
    }
  });
</script>
```

# DOM - Événements



- La méthode *stopPropagation* permet d'empêcher l'appel aux handlers du même événement sur les éléments ancêtres

```
<body>
  <button class="btn">Clic moi</button>
  <div class="output"></div>
  <script type="module">
    const btnElt = document.querySelector('.btn');
    const outputElt = document.querySelector('.output');
    btnElt.addEventListener('click', (event) => {
      event.stopPropagation();
      outputElt.textContent += 'button ';
    });
    btnElt.addEventListener('click', (event) => {
      outputElt.textContent += 'button ';
    });
    document.body.addEventListener('click', (event) => {
      outputElt.textContent += 'body ';
    });
    // au clic du bouton : button button
  </script>
</body>
```

# DOM - Événements



- La méthode `stopImmediatePropagation` permet d'empêcher l'appel aux handlers du même événement sur le même élément

```
<body>
  <button class="btn">Clic moi</button>
  <div class="output"></div>
  <script type="module">
    const btnElt = document.querySelector('.btn');
    const outputElt = document.querySelector('.output');
    btnElt.addEventListener('click', (event) => {
      event.stopImmediatePropagation();
      outputElt.textContent += 'button ';
    });
    btnElt.addEventListener('click', (event) => {
      outputElt.textContent += 'button ';
    });
    document.body.addEventListener('click', (event) => {
      outputElt.textContent += 'body ';
    });
    // au clic du bouton : button
  </script>
</body>
```



**formation.tech**

# APIs Réseaux

# APIs Réseaux - XMLHttpRequest



- Microsoft permet dès 1999 dans IE5 la création de requête AJAX via une bibliothèque appelée XMLHTTP
- Permet à son application Outlook Web Access de récupérer des nouveaux emails sans devoir recharger toute la page
- Mozilla crée une interface appelée XMLHttpRequest compatible avec l'API de Microsoft
- Les requêtes peuvent être asynchrones ou synchrones (absurde car le navigateur serait indisponible le temps de la requête)
- L'API XMLHttpRequest a depuis reçu de nouvelles fonctionnalités en 2011 (requêtes cross-domain, progress events...)
- AJAX : Asynchronous JavaScript And XML



# APIs Réseaux - XMLHttpRequest

- XMLHttpRequest est une fonction constructeur
- La méthode open permet de configurer la méthode HTTP et l'URL
- Des événements comme load, progress ou error peuvent être écoutés
- La méthode send permet d'envoyer la requête

```
// contact.json
{ "firstName": "Bill", "lastName": "Gates" }
```

```
const xhr = new XMLHttpRequest();
xhr.open('GET', 'contact.json');
xhr.addEventListener('load', (event) => {
  const contact = JSON.parse(xhr.response);
  console.log(contact.firstName + ' ' + contact.lastName);
  // Bill Gates
});
xhr.send();
```



# APIs Réseaux - Fetch

- Fetch est un nouvel API plus moderne qu'XMLHttpRequest pour échanger avec le serveur
- L'API est basé sur les promesses
- Fetch n'est pas compatible avec tous les navigateurs :  
[https://developer.mozilla.org/fr/docs/Web/API/Fetch\\_API#Compatibilit%C3%A9\\_Navigateurs](https://developer.mozilla.org/fr/docs/Web/API/Fetch_API#Compatibilit%C3%A9_Navigateurs)
- Github a créé un Polyfill <https://github.com/github/fetch>
- Il existe une implémentation pour Node.js <https://github.com/bitinn/node-fetch>
- Un bibliothèque propose une version isomorphe (compatible Browser et Node.js) <https://github.com/matthew-andrews/isomorphic-fetch>

# APIs Réseaux - Fetch



- La méthode fetch retourne une promesse qui sera résolue en Response
- La Réponse est un stream qui sera lu en asynchrone via des méthodes comme json, blob ou text

```
fetch('contact.json')
  .then((res) => res.json())
  .then((contact) => {
    console.log(contact.firstName + ' ' + contact.lastName);
});
```



# APIs Réseaux - Serveur HTTP Node.js

- Node.js permet la création d'un serveur HTTP rapidement en particulier avec une bibliothèque comme Express
- A installer avec npm :  
npm install express

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

const app = express();

app.get('/contact/1', (req, res) => {
  res.json({ firstName: 'Bill', lastName: 'Gates' });
});

app.listen(3000, () => {
  console.log('Server started');
});
```

# APIs Réseaux - Sécuriser un API REST avec JWT



- JSON Web Token ou JWT permet de générer et de vérifier la signature d'un token simplement
- Le token peut également contenir des informations sur l'utilisateur connecté qui évite d'envoyer une requête vers /users/me comme on le voit sur certains API
- Des bibliothèques permettent de manipuler les tokens JWT
- Côté Node.js (pour générer et vérifier) :  
<https://github.com/auth0/node-jsonwebtoken>
- Côté Navigateur (pour le décoder) :  
<https://github.com/auth0/jwt-decode>

# APIs Réseaux - Sécuriser un API REST avec JWT



- Générer un token JWT

```
app.post('/user/signin', express.json(), (req, res) => {
  if (req.body.username !== user.username || req.body.username !==
  user.username) {
    return res.status(401).json({ error: 'Wrong credentials' });
  }
  const token = jwt.sign({ id: user.id, username: user.username }, secret);
  res.json({ token });
});
```

- Vérifier un token JWT

```
function jwtMiddleware(req, res, next) {
  const token = req.headers.authorization;
  try {
    jwt.verify(token, secret);
  } catch(err) {
    return res.status(401).json({ error: 'Unauthorized' });
  }
  return next();
}
```

# APIs Réseaux - WebSocket



- Les requêtes AJAX avec XHR ou Fetch permettent un échange via le protocole HTTP
- HTTP jusqu'à sa version 1.1 (la plus répandue actuellement) est à l'initiative du client (Client Pool)
- HTTP 2.0 permet la mise en place d'événement serveur (ou Server Push)
- Les WebSockets elles, permettent d'établir un canal de communication permanent entre le client et le serveur, leur permettant d'échanger dans les 2 sens



# APIs Réseaux - WebSocket

- Exemple avec le echo de [websocket.org](https://websocket.org) (retourne le message envoyé)

```
const ws = new WebSocket('wss://echo.websocket.org');
ws.addEventListener('open', () => {
    console.log('Connecté');
    ws.send('Bonjour');
    ws.send('Bye');
    setTimeout(() => {
        ws.close();
    }, 1000);
});
ws.addEventListener('close', () => {
    console.log('Déconnecté')
});
ws.addEventListener('message', (event) => {
    console.log('Message : ' + event.data);
});
// Connecté
// Message : Bonjour
// Message : Bye
// Déconnecté
```



# APIs Réseaux - WebSocket

- Côté Node.js plusieurs implémentations de serveur WebSocket existent
  - `ws`
  - [socket.io](#)
- Exemple : un server echo

```
const { Server } = require('ws');

const server = new Server({ port: 8080 });

server.on('connection', (ws) => {
  ws.on('message', (message) => {
    ws.send(message);
  });
});
```



# APIs Réseaux - EventSource

- Si la communication ne se fait que dans le sens Serveur → Client on peut utiliser un nouvel API : EventSource

```
const es = new EventSource('http://localhost:3000/event-stream');
es.addEventListener('open', () => {
  console.log('Connecté');
});
es.addEventListener('message', (event) => {
  console.log(event.data);
});
es.addEventListener('close', () => {
  console.log('Déconnecté');
});
```

# APIs Réseaux - EventSource



- Côté serveur, on va configurer une réponse HTTP qui ne se termine jamais

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

const app = express();

app.get('/event-stream', (req, res) => {
  res.writeHead(200, {
    'Content-Type': 'text/event-stream',
    'Cache-Control': 'no-cache',
    'Connection': 'keep-alive',
    'Access-Control-Allow-Origin': '*',
  });
  res.write('\n');

  let id = 0;
  const timeout = setInterval(() => {
    res.write(`id: ${++id}\n`);
    res.write(`data: ${Date.now()}\n\n`);
  }, 1000);

  req.on('close', () => clearInterval(timeout));
});

app.listen(3000);
```



**formation.tech**

# APIs de Stockage



# APIs de Stockage - Introduction

- Historiquement, seuls les cookies permettent le stockage de données persistantes dans le navigateur
- Aujourd'hui :
  - Cookies
  - Local Storage
  - Session Storage
  - WebSQL (déprécié)
  - IndexedDB
  - FileSystem (non standard)
  - Cache



# APIs de Stockage - Introduction

- Synchrone vs Asynchrone  
Certains API comme localStorage sont synchrone d'autres comme IndexedDB asynchrones
- Structure de données  
On retrouve des APIs clé/valeurs et d'autres plus structurés
- Transactions  
Certains API supporte les transactions (empêche l'écriture si une parmi un ensemble échoue)
- Comparaison  
<https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/#comparison>



# APIs de Stockage - Same-origin policy

- L'origin est la combinaison de :
  - Schéma d'URI
  - Domaine
  - Eventuellement le port
- Exemples :  
<https://www.google.fr>  
<http://localhost:3000>
- Les API de Stockage impriment la same-origin policy, c'est à dire que les valeurs sont associées à une origin en particulier

# APIs de Stockage - Cookies



- Les cookies peuvent être créés par le serveur et envoyé automatiquement
- Côté on utilise la propriété `document.cookie` en lecture/écriture
- Il est conseillé d'utiliser une bibliothèque comme js-cookie  
<https://github.com/js-cookie/js-cookie>
- Exemple de valeur de `document.cookie` (Github) :  
`_ga=GA1.2.2043353612.1547462011; _octo=GH1.1.515337093.1547462012;  
tz=Europe%2FParis; has_recent_activity=1; _gat=1`
- Sans bibliothèque il faudrait écrire quelque chose comme :

```
const cookies = Array.from(document.cookie.matchAll(/(\w+)=([^\;]+)\;/g))
    .map(([key,value]) => ({key, value}));
```

```
// [
//   {key: "_ga", value: "1"},
//   {key: "_octo", value: "2"},
//   {key: "tz", value: "s"},
//   {key: "has_recent_activity", value: "1"},
//   {key: "_gat", value: "1"}
// ]
```

# APIs de Stockage - Local Storage et Session Storage



- Local Storage et Session Storage ont le même API
- Permettent comme les cookies la manipulation synchrone de données clés/valeurs
- Le localStorage est un espace de stockage associé au navigateur et qui persiste après sa fermeture
- Le sessionStorage est un espace de stockage associé à l'onglet d'un navigateur et est vidé une fois l'onglet fermé

```
1 <script>
2 localStorage.
```

The screenshot shows a code editor with two lines of JavaScript. The second line starts with 'localStorage.' followed by a red squiggly underline under the period. A tooltip or dropdown menu is open below the cursor, listing six methods: 'clear', 'getItem', 'key', 'length', 'removeItem', and 'setItem'. Each method is preceded by a purple cube icon.

```
1 <script>
2 sessionStorage.
```

The screenshot shows a code editor with two lines of JavaScript. The second line starts with 'sessionStorage.' followed by a red squiggly underline under the period. A tooltip or dropdown menu is open below the cursor, listing six methods: 'clear', 'getItem', 'key', 'length', 'removeItem', and 'setItem'. Each method is preceded by a purple cube icon.

# APIs de Stockage - Local Storage et Session Storage



- On utilise souvent le localStorage pour stocker le token JWT

```
const token = 'eyJhbGciOiJIUzI...';
localStorage.setItem('token', token);

console.log(localStorage.getItem('token')); // 'eyJhbGciOiJIUzI...'
```

# APIs de Stockage - IndexedDB



- Le localStorage est adapté au stockage de petites valeurs (tokens, locale, préférences, ...)
- Pour des valeurs plus conséquentes il faudrait privilégier IndexedDB qui :
  - est asynchrone
  - supporte les transactions
  - offre limite de stockage est bien plus grand, quelques Go contre quelques Mo (la limite dépend des navigateurs et de l'espace disque disponible)
- L'API est un peu complexe à utiliser, on pourrait privilégier des bibliothèques comme
  - idb <https://github.com/jakearchibald/idb>
  - localForage <https://localforage.github.io/localForage/>



# APIs de Stockage - IndexedDB

- Exemple

```
const openRequest = indexedDB.open('address-book', 1);

openRequest.addEventListener('upgradeneeded', () => {
  const db = openRequest.result;
  const store = db.createObjectStore('contacts', { autoIncrement: true });
  store.put({firstName: 'Bill', lastName: 'Gates'});
  store.put({firstName: 'Steve', lastName: 'Jobs'});
})

openRequest.addEventListener('success', (event) => {
  const db = openRequest.result;
  const request = db.transaction('contacts').objectStore('contacts').getAll();

  request.addEventListener('success', (event) => {
    console.log(request.result);
    // [
    //   { firstName: 'Bill', lastName: 'Gates' },
    //   { firstName: 'Steve', lastName: 'Jobs' }
    // ]
  })
});
```



**formation.tech**

# Workers

# Workers - Introduction



- › Les navigateur exécute le code JavaScript dans 1 seul thread
- › Ce thread fait également le rendu de la page
- › Lorsqu'un traitement JavaScript devient trop lourd, bloquer le thread empêchera le navigateur de dessiner la page, rendant ainsi toute interaction impossible
- › Pour éviter cela, on peut utiliser un Worker qui exécutera le code JavaScript lourd dans un thread séparé et communiquera avec le thread principal via des événements



# Workers - Exemple

- Exemple : recherche de nombre premiers >  $2^{52}$

```
// prime-nbs.js
for (let nb = 2 ** 52; nb < Number.MAX_SAFE_INTEGER; nb++) {
    let isPrime = true;

    for (let i = 2; i < Math.sqrt(nb) + 1; i++) {
        if (nb % i === 0) {
            isPrime = false;
            break;
        }
    }

    if (isPrime) {
        postMessage(nb);
    }
}
```

```
const worker = new Worker('prime-nbs.js');
let date = Date.now();

worker.addEventListener('message', (event) => {
    console.log(event.data);
    console.log('Trouvé en : ' + ((Date.now() - date) / 1000) + 's');
    date = Date.now();
    // 4503599627370517
    // Trouvé en : 0.603s
    // ...
});
```



**formation.tech**

# DevTools

# DevTools - Introduction



- A partir 2005 les premiers outils de développements apparaissent dans les navigateurs, d'abord sous forme d'extension (Firebug)
- Les DevTools des navigateurs modernes :
  - Inspecteur du DOM
  - Déboggage JavaScript
  - Temps de chargements
  - Profilage des performances
  - Consommation mémoire
  - Consultation des données stockées
  - Audits
  - ...

# DevTools - Chrome DevTools



- Les DevTools de Chrome sont aujourd'hui les outils les plus aboutis
- Documentation  
<https://developers.google.com/web/tools/chrome-devtools/>
- Suivre régulièrement les (nombreuses) nouveautés apportées par Chrome à chaque nouvelle version des DevTools  
<https://developers.google.com/web/updates/2019/>

# DevTools - Chrome DevTools



## ▶ Inspecteur du DOM

The screenshot shows the Chrome DevTools interface with the 'Elements' tab selected. The left pane displays the DOM tree for a page titled 'DevTools - chrome-search://local-ntp/local-ntp.html'. The right pane shows the 'Styles' tab of the Elements panel, listing the CSS rules applied to the selected element.

**DOM Tree (Elements Tab):**

```
<!doctype html>
<html lang="fr" darkmode="false">
  <!-- TODO(dbeam): dir="ltr"? -->
  <!-- Copyright 2015 The Chromium Authors. All rights reserved.
      Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. -->
  ><head>...</head>
... <body class="light-chip alternate-fakebox show-fakebox-icon mac initied" style="background: rgb(255, 255, 255);"> == $0
    <div id="custom-bg" style="opacity: 0;"></div>
    <!-- Container for the OneGoogleBar HTML. -->
    ><div id="one-google" class="show-element">...
    </div>
    ><script type="text/javascript">...</script>
    ><div id="ntp-contents" class="default-theme">...
    </div>
    ><dialog div id="edit-bg-dialog">...</dialog>
    ><dialog id="ddlsd">...</dialog>
    ><dialog id="ba_SEL_menu" class="customize-...
...   body
```

**Styles Tab (Elements Panel):**

Style	Source
element.style { background: □rgb(255, 255, 255); }	local-ntp.css:84
body.initied { display: block; }	local-ntp.css:75
body { background-attachment: fixed !important; cursor: default; display: none; font-size: small; margin: □0; min-height: 100%; }	user agent stylesheet
body { display: block; margin: □8px; }	
Inherited from html	

# DevTools - Chrome DevTools



## ► Déboggage JavaScript

The screenshot shows the Chrome DevTools interface with the "Sources" tab selected. The left sidebar shows a file tree for a "local-ntp" project, with "doodles.js" currently open. The code editor displays the following JavaScript code:

```
// Copyright 2018 The Chromium Authors. All
// Use of this source code is governed by a
// found in the LICENSE file.
const doodles = {};
doodles.numDdllogResponsesReceived = 0;
doodles.lastDdllogResponse = '';
doodles.onDdllogResponse = null;
doodles.ei = null;
/** * Enum for classnames. * @enum {string} * @const */
doodles.CLASSES = {
  FADE: 'fade',           // Enables opacity
  SHOW_LOGO: 'show-logo' // Marks logo/doo
}
```

The line `doodles.onDdllogResponse = null;` is highlighted with a blue selection bar, indicating it is the current line of execution. The status bar at the bottom of the code editor shows "Line 11, Column 1".

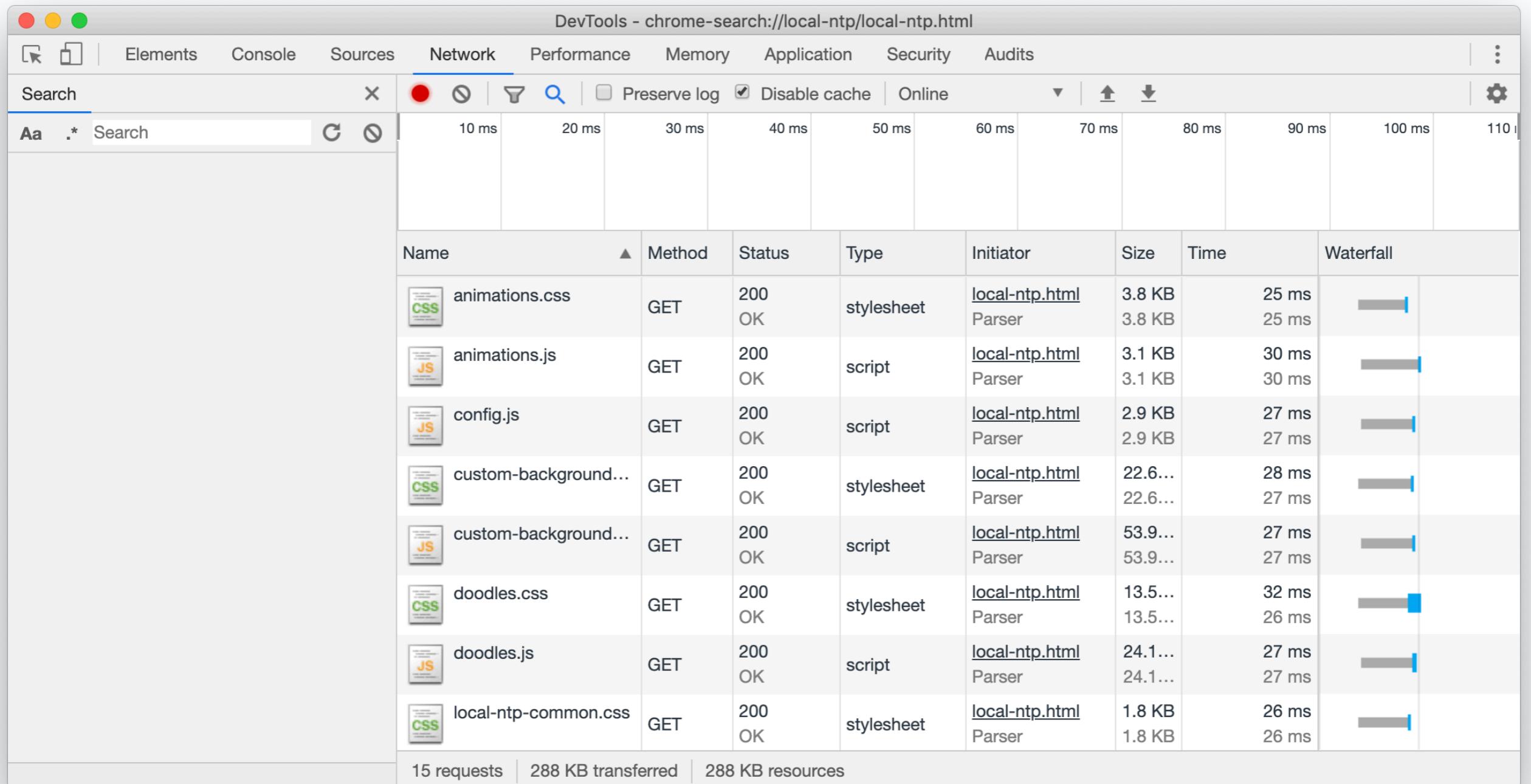
The right panel contains the DevTools sidebar with the following sections:

- Paused on breakpoint** (highlighted in yellow)
- Watch
- Call Stack
  - (anonymous) doodles.js:11
- Scope
- Script
  - BACKGROUND\_CUSTOMIZATION\_LOG\_TYPE: {NTP...}
  - animations: {CLASSES: {...}, RIPPLE\_DURATI...}
  - customBackgrounds: {KEYCODES: {...}, IDS: ...}
  - doodles: {numDdllogResponsesReceived: 0,...}
  - fadeInImageTile: f (tile, imageUrl, coun...
  - loadTile: f (tile, imageData, countLoad)
  - ntpApiHandle: undefined
- Global (Window)
- Breakpoints
  - doodles.js:11  
doodles.onDdllogResponse = null;

# DevTools - Chrome DevTools



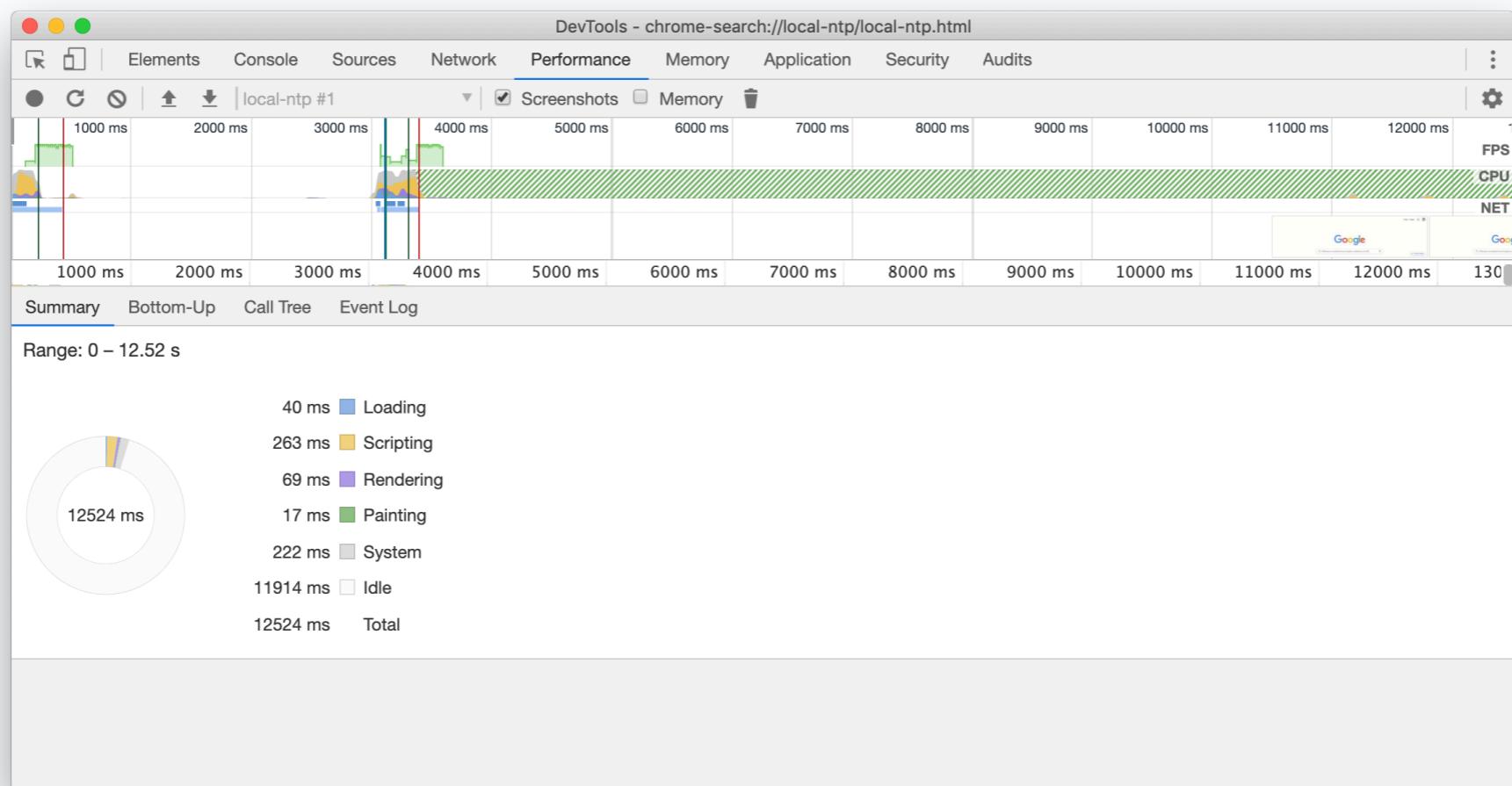
## ► Temps de chargements



# DevTools - Chrome DevTools



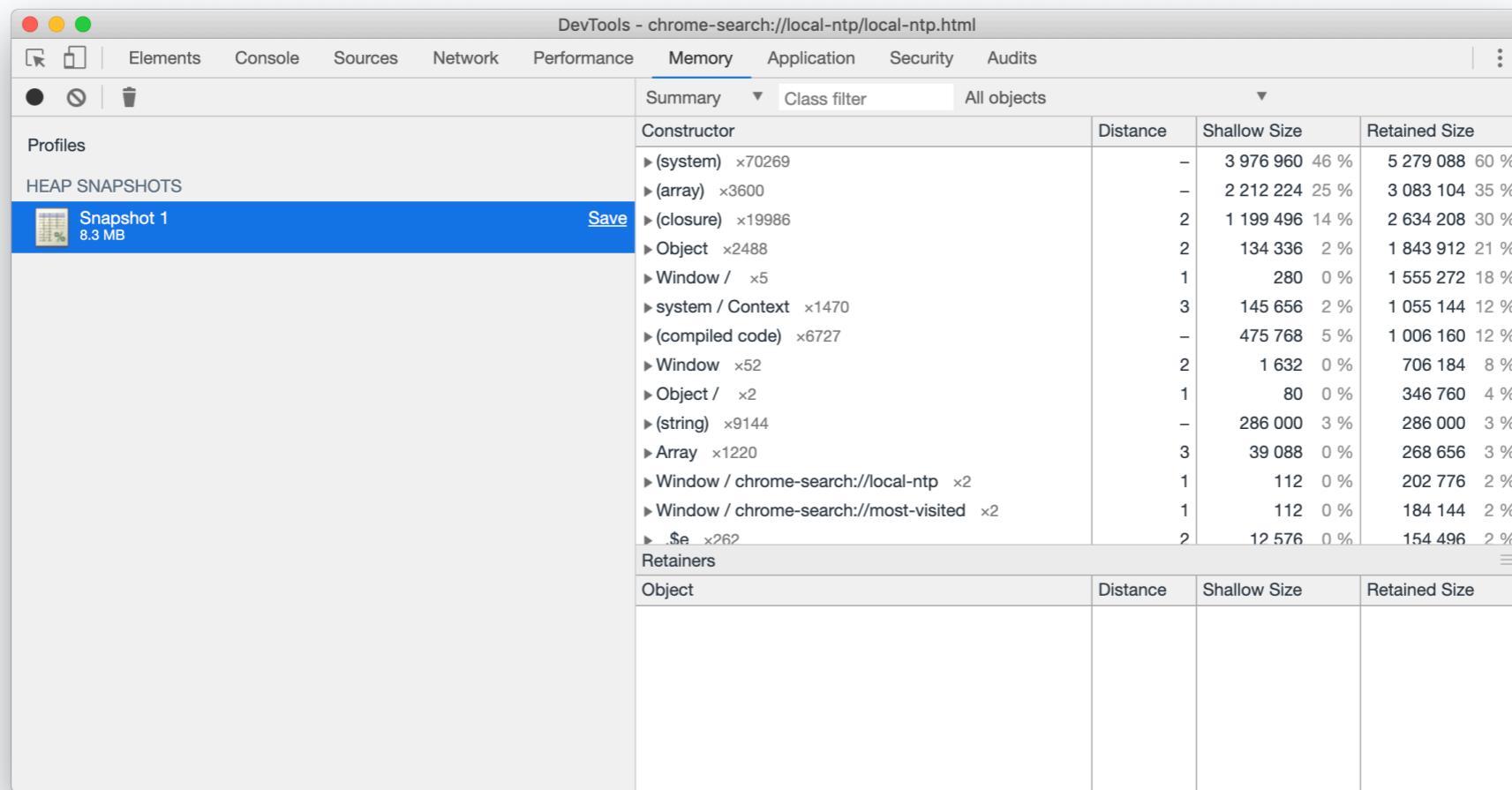
- Profilage des performances
- permet d'identifier les problèmes de performances en analysant la pile d'appels de fonction
- permet d'identifier des fuites mémoires dans le temps



# DevTools - Chrome DevTools



- Consommation mémoire
- Permet de consulter le contenu de la mémoire et d'identifier les objets qui ne sont plus utilisés



The screenshot shows the Chrome DevTools Memory tab interface. At the top, there are tabs for Elements, Console, Sources, Network, Performance, Memory (which is selected), Application, Security, and Audits. Below the tabs, there's a toolbar with icons for profiles, heap snapshots, and a save button. The main area is divided into two sections: 'Profiles' on the left and 'HEAP SNAPSHOTS' on the right. Under 'HEAP SNAPSHOTS', a blue bar indicates 'Snapshot 1' is active, with a size of 8.3 MB and a 'Save' button. The main content area displays a table titled 'Summary' with columns for 'Constructor', 'Distance', 'Shallow Size', and 'Retained Size'. The table lists various object types and their memory usage details. A second table titled 'Retainers' is also present but currently empty.

Constructor	Distance	Shallow Size	Retained Size
► (system) ×70269	-	3 976 960 46 %	5 279 088 60 %
► (array) ×3600	-	2 212 224 25 %	3 083 104 35 %
► (closure) ×19986	2	1 199 496 14 %	2 634 208 30 %
► Object ×2488	2	134 336 2 %	1 843 912 21 %
► Window / ×5	1	280 0 %	1 555 272 18 %
► system / Context ×1470	3	145 656 2 %	1 055 144 12 %
► (compiled code) ×6727	-	475 768 5 %	1 006 160 12 %
► Window ×52	2	1 632 0 %	706 184 8 %
► Object / ×2	1	80 0 %	346 760 4 %
► (string) ×9144	-	286 000 3 %	286 000 3 %
► Array ×1220	3	39 088 0 %	268 656 3 %
► Window / chrome-search://local-ntp ×2	1	112 0 %	202 776 2 %
► Window / chrome-search://most-visited ×2	1	112 0 %	184 144 2 %
► .\$e ×262	2	12 576 0 %	154 496 2 %

Object	Distance	Shallow Size	Retained Size



# DevTools - Chrome DevTools

- Consultation des données stockées

DevTools - localhost:8080/01-hello.html

Elements Console Sources Network Performance Memory Application Security Audits

Key	Value
Drift.Targeting.firstVisit	1548452260
Drift.Targeting.currentPageViewStarted	1553348301
Drift.Targeting.previousSessionStartedAt	1553339750
Drift.Targeting.currentReferrer	""
staytuned-cid	40315c1a-b0b3-4c00-ba66-ee5572ffccf1
Drift.Targeting.previousPage	"localhost:8080/formations/php/php-mysql"
algoliasearch-client-js	{"Y1R97722NF":{"hostIndexes":{"read":0,"write":0},"timeoutMulti...
Drift.Targeting.referrerDomain	""
Drift.Targeting.currentSessionStartedAt	1553347876
Drift.Targeting.numberOfVisits	207
Drift.Targeting.previousSessionEndedAt	1553345827
loglevel:webpack-dev-server	INFO
Drift.Targeting.lastVisit	1553348301

Cache

- Cache Storage
- Application Cache

Background Services

- Background Fetch
- Background Sync

Frames

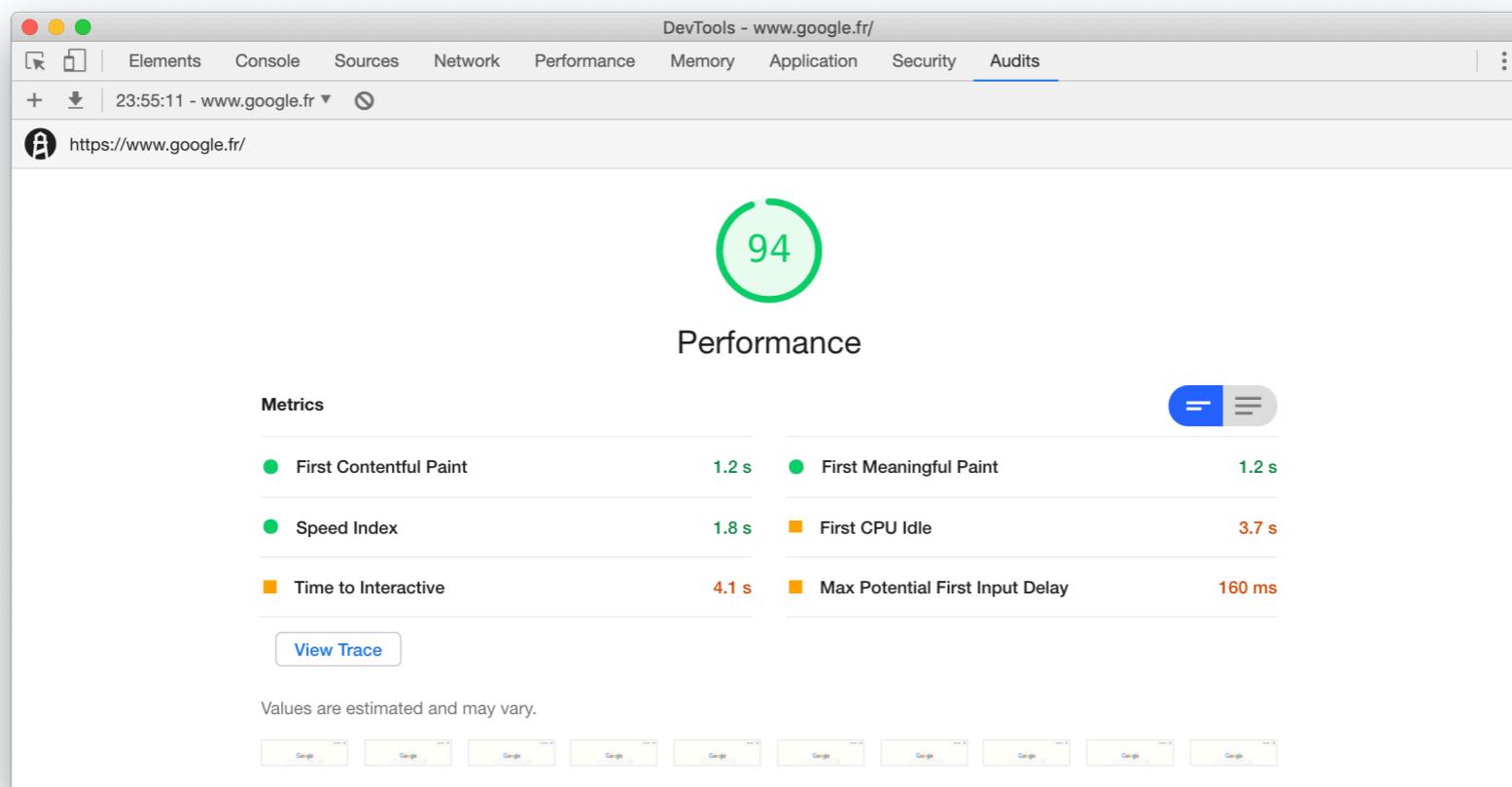
- top

Line 1, Column 1



# DevTools - Chrome DevTools

- Audits (anciennement LightHouse)
- Génère un audit avec des conseils sur la performance, le SEO, l'accessibilité...





# DevTools - Firefox Developer Tools

- Firefox propose ses propres outils appelés Developer Tools  
<https://developer.mozilla.org/en-US/docs/Tools>

The screenshot shows the Firefox Developer Tools open in a browser window. The tabs at the top are Inspector, Console, Debugger, Performance, Network, Storage, and Accessibility. The Inspector tab is active, displaying the HTML pane with code snippets and the CSS pane with a list of CSS rules. The Layout tab is selected in the bottom navigation bar, and its content is shown in the Layout pane, which displays a visual representation of a box model with margins, borders, and padding.

**Inspector Tab:**

- HTML Pane: Shows the DOM structure with elements like <body id="home" class="">, <main id="content" role="main">, and <footer id="nav-footer" class="nav-footer">.
- CSS Pane: Shows a list of CSS rules, such as element { inline } and body { line-height: 1.6; font-family: Arial, sans-serif; letter-spacing: -.00278rem; font-size: 10px; font-size: 1.25rem; -webkit-font-smoothing: antialiased; -moz-osr font-smoothing: antialiased; }.

**Layout Tab:**

- Layout Pane: Visual representation of the box model for an element, showing margin (0), border (0), padding (0), and content (Layout Pane).
- Box Model Properties: Shows properties like box-sizing (content-box), display (block), and float (none).



# DevTools - Node.js

- Pour déboguer un script Node.js on peut utiliser
  - Un débogueur en ligne de commande
  - Les DevTools de Chrome
  - Le débogueur d'un IDE comme Visual Studio Code ou WebStorm



# DevTools - Node.js

- En ligne de commande :

```
node inspect fichier.js
```

```
ES6 — node < node inspect Object.assign.js — 81x17
MacBook-Pro-de-Romain:ES6 roman$ node inspect Object.assign.js
< Debugger listening on ws://127.0.0.1:9229/e1c56f99-9bd5-4df1-b2ca-25e6c36c7e54
< For help, see: https://nodejs.org/en/docs/inspector
< Debugger attached.

Break on start in Object.assign.js:1
> 1 const obj = {
  2   firstName: 'Romain',
  3   address: {
debug> next
break in Object.assign.js:8
  6 };
  7
> 8 const copy = Object.assign({}, obj);
  9 console.log(copy === obj); // false
 10 console.log(copy.address === obj.address); // true
debug>
```

# DevTools - Node.js



- Avec les DevTools de Chrome, lancer la commande :  
node inspect --inspect-brk fichier.js
  - Puis aller à l'URL chrome://inspect dans Chrome

The screenshot shows the Chrome DevTools interface. The left sidebar is titled "DevTools" and lists various sections: Devices, Pages, Extensions, Apps, Shared workers, Service workers, and Other. Under "Devices", there are checkboxes for "Discover USB devices" and "Discover network targets", both of which are checked. Below these are buttons for "Port forwarding..." and "Configure...". A link "Open dedicated DevTools for Node" is also present. The main content area is titled "Inspect with Chrome DevTools" and shows the URL "chrome://inspect/#devices". The top navigation bar has tabs for Profiler, Console, Sources (which is selected), and Memory. The Sources tab displays the code for "Object.assign.js" with line numbers 1 and 2 visible. The code is as follows:

```
1 const obj = { obj: {firstName: "Romain", address: {...}}  
2   firstName: 'Romain',  
3   address: {  
4     street: "1600 Amphitheatre Pkwy",  
5     zip: 94035, city: "Mountain View", state: "CA", country: "US"  
6   }  
7 }  
8  
9 console.log(obj); // true  
10  
11 if (false) {  
12   console.log(obj.address); // true  
13 }  
14  
15 let obj = {  
16   address: {  
17     street: "1600 Amphitheatre Pkwy",  
18     zip: 94035, city: "Mountain View", state: "CA", country: "US"  
19   }  
20 }  
21  
22 console.log(obj); // false  
23  
24 if (false) {  
25   console.log(obj.address); // false  
26 }  
27  
28 const obj = {  
29   obj: {firstName: "Romain", address: {...}}  
30   firstName: 'Romain',  
31   address: {  
32     street: "1600 Amphitheatre Pkwy",  
33     zip: 94035, city: "Mountain View", state: "CA", country: "US"  
34   }  
35 }  
36  
37 console.log(obj); // true  
38  
39 if (false) {  
40   console.log(obj.address); // true  
41 }  
42  
43 let obj = {  
44   address: {  
45     street: "1600 Amphitheatre Pkwy",  
46     zip: 94035, city: "Mountain View", state: "CA", country: "US"  
47   }  
48 }  
49  
50 console.log(obj); // false  
51  
52 if (false) {  
53   console.log(obj.address); // false  
54 }  
55  
56 const obj = {  
57   obj: {firstName: "Romain", address: {...}}  
58   firstName: 'Romain',  
59   address: {  
60     street: "1600 Amphitheatre Pkwy",  
61     zip: 94035, city: "Mountain View", state: "CA", country: "US"  
62   }  
63 }  
64  
65 console.log(obj); // true  
66  
67 if (false) {  
68   console.log(obj.address); // true  
69 }  
70  
71 let obj = {  
72   address: {  
73     street: "1600 Amphitheatre Pkwy",  
74     zip: 94035, city: "Mountain View", state: "CA", country: "US"  
75   }  
76 }  
77  
78 console.log(obj); // false  
79  
80 if (false) {  
81   console.log(obj.address); // false  
82 }  
83  
84 const obj = {  
85   obj: {firstName: "Romain", address: {...}}  
86   firstName: 'Romain',  
87   address: {  
88     street: "1600 Amphitheatre Pkwy",  
89     zip: 94035, city: "Mountain View", state: "CA", country: "US"  
90   }  
91 }  
92  
93 console.log(obj); // true  
94  
95 if (false) {  
96   console.log(obj.address); // true  
97 }  
98  
99 let obj = {  
100   address: {  
101     street: "1600 Amphitheatre Pkwy",  
102     zip: 94035, city: "Mountain View", state: "CA", country: "US"  
103   }  
104 }  
105  
106 console.log(obj); // false  
107  
108 if (false) {  
109   console.log(obj.address); // false  
110 }
```

The right panel shows the "Call Stack" for the current script. The stack trace includes:

- (anonymous) at Object.assign.js:8
- Module.\_compile internal/module.../loader.js:865
- Module.\_extensions..js internal/module.../loader.js:879
- Module.load internal/module.../loader.js:731
- Module.\_load internal/module.../loader.js:644
- Module.runMain internal/module.../loader.js:931
- (anonymous) internal/main/r...n\_module.js:17

The "Scope" section shows the local variables:

- copy: undefined
- deepCopy: undefined
- exports: {}
- module: Module {id: ".", path: "/Users/romain/..."}
- obj: {firstName: "Romain", address: {...}}
- require: f require(path)
- this: Object {\_\_dirname: "/Users/romain/www/Learning/JavaScript", \_\_filename: "/Users/romain/www/Learning/JavaScript/Object.assign.js"}

The "Global" section shows the global variable:

- global

The "Breakpoints" section is currently empty.



# DevTools - Node.js

- › Dans un IDE

The screenshot shows the Node.js DevTools interface within a dark-themed IDE. The top bar displays the title "Object.assign.js — ES6". The left sidebar contains icons for file, search, connections, security, and settings, along with sections for "VARIABLES" (Local variables like `this`, `\_\_dirname`, `\_\_filename`, `copy`, and `deepClone`), "WATCH" (empty), and "CALL STACK" (Paused on break, showing stack frames for `Module.\_compile` and `loader.js`).

The main area shows a code editor with the following JavaScript code:

```
JS Object.assign.js > [8] copy
1 const obj = {
2   firstName: 'Romain',
3   address: {
4     city: 'Paris'
5   }
6 };
7
8 const copy = Object.assign({}, obj);
9 console.log(copy === obj); // false
10 console.log(copy.address === obj.address); // true
11
12 const deepClone = function(obj) {
13   let clone = Object.assign({}, obj);
```

The line `const copy = Object.assign({}, obj);` is highlighted with a yellow background, indicating it is the current line of execution.

The bottom right pane shows the "DEBUG CONSOLE" output:

```
Debugger listening on ws://127.0.0.1:31772/9811094a-7036-4f73-939e-974f2824cee8
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.
```

The bottom status bar indicates the line is at "Ln 8, Col 14" and the file type is "JavaScript". There are also icons for Prettier, smiley face, and bell.

# DevTools - Gérer les logs en production



- Pour pouvoir désactiver les logs (console.log...) il faudra passer par une bibliothèque
- Une des plus utilisé debug (50 000 000 de téléchargement par semaine)  
<https://github.com/visionmedia/debug#readme>
- Avec debug pour que les logs s'affichent, il faut créer une variable d'environnement côté Node.js ou remplir une clé debug du localStorage côté navigateur

```
const debug = require('debug')('http')
const http = require('http')
const name = 'My App';

debug('booting %o', name);

const server = http.createServer((req, res) => {
  debug(req.method + ' ' + req.url);
  res.end('Hello');
});

server.listen(3000, () => {
  debug('listening');
});
```



**formation.tech**

# Modules ECMAScript

# Modules ECMAScript - Introduction



- JavaScript à sa conception
  - Objectif : créer des interactions côté client, après chargement de la page
  - Exemples de l'époque :
    - Menu en rollover (image ou couleur de fond qui change au survol)
    - Validation de formulaire
- JavaScript aujourd'hui
  - Applications front-end, back-end, en ligne de commande, de bureau, mobiles...
  - Applications pouvant contenir plusieurs centaines de milliers de lignes de codes (Front-end de Facebook > 1 000 000 LOC)
  - Il faut faciliter le travail collaboratif, en plusieurs fichiers et en limitant les risques de conflit

# Modules ECMA Script - Introduction

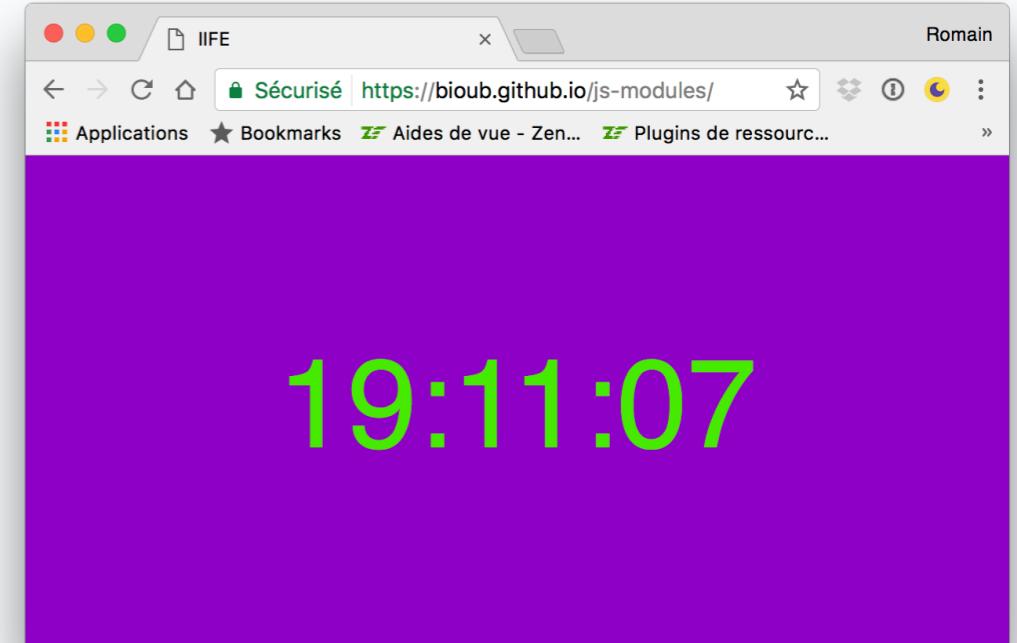
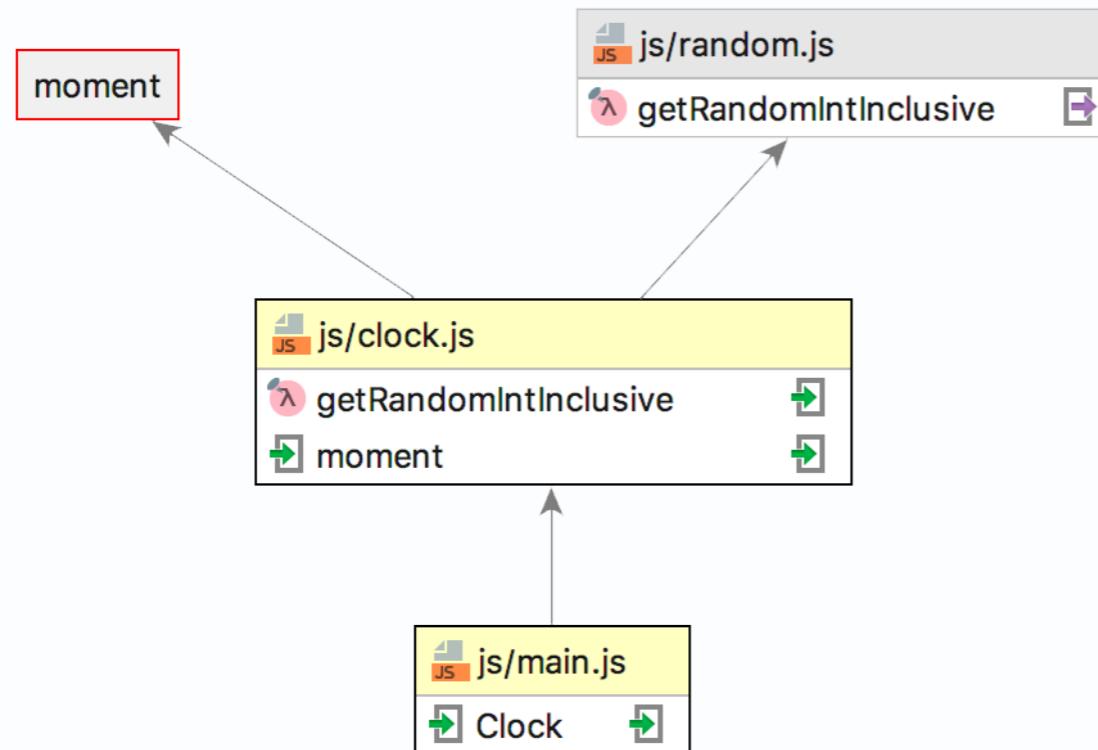


- Objectifs d'un module JavaScript
  - Créer une portée au niveau du fichier
  - Permettre l'export et l'import d'identifiants (variables, fonctions...) entre ces fichiers qui auront désormais leur propre portée
- Principaux systèmes existants
  - IIFE / Function Wrapper
  - CommonJS
  - AMD
  - UMD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

# Modules ECMA Script - Introduction



- Exemple utilisé pour la suite



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

# Module ECMAScript - Concepts



- Portée de modules
  - Sans module la portée d'une fonction ou d'une variable déclarée dans un fichier serait globale.
  - Avec les modules une fonction sera locale au fichier.
- Import / Export
  - Une fonction ou un objet pouvant servir dans un autre fichier il faudra l'exporter.
  - Cela va créer l'API public du fichier (accessible de l'extérieur).
  - Un autre fichier devra importer les fonctions utilisées
- Mode Strict
  - Les modules ECMAScript sont par défaut en mode strict, il n'est donc pas nécessaire d'écrire '`use strict`'; en début de fichier.



# Module ECMAScript - Export

- Pour exporter une variable ou une fonction on utilise le mot clé export

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

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

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

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



# Module ECMAScript - Export

- Il est également possible d'exporter en une seule fois en fin de fichier

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

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

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

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

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

# Module ECMAScript - Import



- Pour importer on utilise le mot clé import, associé à des accolades et le nom du fichier (l'extension est optionnelle)
- Lorsque que le fichier fait partie du projet, il est obligatoire de préfixer le fichier par ./ ou ../
- Les modules ECMAScript ne peuvent être importée que statiquement en début de fichier. Pour des imports dynamiques il faut utiliser les modules CommonJS ou Dynamic Import (ESNext)

```
import { getRandomIntInclusive } from './random';

class Clock {
    // ...

    update() {
        let r = getRandomIntInclusive(0, 255);
        let g = getRandomIntInclusive(0, 255);
        let b = getRandomIntInclusive(0, 255);
        // ...
    }
    // ...
}
```

# Module ECMAScript - Tree Shaking



- Les imports étant statiques, des bundlers (bibliothèques de build) comme webpack ou Rollup peuvent analyser le code et éliminer du build les exports non importés
- Le build final ressemblera ainsi à :

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

class Clock {
  // ...

  update() {
    let r = getRandomIntInclusive(0, 255);
    let g = getRandomIntInclusive(0, 255);
    let b = getRandomIntInclusive(0, 255);
    // ...
  }
  // ...
}
```



# Module ECMAScript - Export/Import par défaut

- Il est possible de définir un export par défaut lorsqu'on a qu'une seule valeur à importer ou une valeur principale à importer
- Pour exporter on ajoute le mot clé *default*

```
export default class Clock {  
    // ...  
}
```

- Pour importer il faudra ne pas utiliser d'accolades

```
import Clock from './clock';  
  
let clockElt = document.querySelector('.clock');  
let clock = new Clock(clockElt);  
clock.start();
```

- Certains développeurs conseillent d'éviter les exports par défaut :  
<https://basarat.gitbooks.io/typescript/docs/tips/defaultIsBad.html>



# Module ECMAScript - Imports avancés

- On peut renommer un import, par exemple dans le cas où 2 identifiants auraient le même nom :

```
import { render as ReactDOM } from 'react-dom';
import { App } from './App';

ReactDOM(<App />, document.getElementById('root'));
```

- On peut également importer tous les exports dans un objet :

```
// serviceWorker.js
export function register(config) {
  // ...
}

export function unregister() {
  // ...
}
```

```
import * as serviceWorker from './serviceWorker';

serviceWorker.unregister();
```



**formation.tech**

# TypeScript



# TypeScript - Introduction

- TypeScript : JavaScript + Typage statique
  - TypeScript est un langage créé par Microsoft, construit comme un sur-ensemble d'ECMAScript
  - Pour pouvoir exécuter le code il faut le transformer en JavaScript avec un compilateur
  - A quelques exceptions près et selon la configuration, le JavaScript est valide en TypeScript
  - Le principal intérêt de TypeScript est l'ajout d'un typage statique



# TypeScript - Installation

- Installation
  - `npm install -g typescript`
- Création d'un fichier de configuration
  - `tsc --init`
- Compilation
  - `tsc`



# TypeScript - Typage statique

- Le principal intérêt de TypeScript est l'introduction d'un typage statique

```
const lastName: string = 'Bohdanowicz';
const age: number = 32;
const isTrainer: boolean = true;
```

- Types basiques :
  - boolean*
  - number*
  - string*



# TypeScript - Typage statique

- Avantages
  - Complétion

```
const firstName: string = 'Romain';

function hello(firstName: string): string {
    return `Hello ${firstName}`;
}

m ↵ charAt(pos: number)
m ↵ charCodeAt(index: number)
m ↵ concat(... strings: string)
m ↵ indexOf(searchString: string,
```

- Détection des erreurs

```
const firstName: string = 'Romain';

function hello(firstName: string): string {
    return `Hello ${firstName}`;
}

hello({
    firstName: 'Romain',
});
```



# TypeScript - Typage statique

- › Tableaux

```
const firstNames: string[] = ['Romain', 'Edouard'];
const colors: Array<string> = ['blue', 'white', 'red'];
```

- › Tuples

```
const email: [string, boolean] = ['romain.bohdanowicz@gmail.com', true];
```

- › Enum

```
enum Choice {Yes, No, Maybe}

const c1: Choice = Choice.Yes;
const choiceName: string = Choice[1];
```

- › Never

```
function error(message: string): never {
    throw new Error(message);
}
```



# TypeScript - Typage statique

- Any

```
let anyType: any = 12;
anyType = "now a string string";
anyType = false;
anyType = {
  firstName: 'Romain'
};
```

- Void

```
function withoutReturn(): void {
  console.log('Do someting')
}
```

- Null et undefined

```
let u: undefined = undefined;
let n: null = null;
```



# TypeScript - Assertion de type

- Le compilateur ne peut pas toujours déterminer le type adéquat :

```
const formElt = document.querySelector('#myForm');
const url = formElt.action; // error TS2339: Property 'action' does not exist on
                           type 'Element'.
```

- Il faut alors lui préciser, 3 syntaxes possibles

```
let formElt = <HTMLFormElement> document.querySelector('#myForm');
const url = formElt.action;
```

```
let formElt = document.querySelector<HTMLFormElement>('#myForm');
const url = formElt.action;
```

```
let formElt = document.querySelector('#myForm') as HTMLFormElement;
const url = formElt.action;
```



# TypeScript - Inférence de type

- TypeScript peut parfois déterminer automatiquement le type :

```
const title = 'First Names';
console.log(title.toUpperCase());

const names = ['Romain', 'Edouard'];
for (let n of names) {
  console.log(n.toUpperCase());
}
```



# TypeScript - Interfaces

- Pour documenter un objet on utilise une interface
  - Anonyme

```
function helloInterface(contact: {firstName: string}) {  
    console.log(`Hello ${contact.firstName.toUpperCase()}`);  
}
```

- Nommée

```
interface ContactInterface {  
    firstName: string;  
}  
  
function helloNamedInterface(contact: ContactInterface) {  
    console.log(`Hello ${contact.firstName.toUpperCase()}`);  
}
```



# TypeScript - Interfaces

- Les propriétés peuvent être :
  - optionnelles (ici *lastName*)
  - en lecture seule, après l'initialisation (ici *age*)
  - non déclarées (avec les crochets)

```
interface ContactInterface {  
    firstName: string;  
    lastName?: string;  
    readonly age: number;  
    [propName: string]: any;  
}  
  
function helloNamedInterface(contact: ContactInterface) {  
    console.log(`Hello ${contact.firstName.toUpperCase()}`);  
}
```



# TypeScript - Classes

- Quelques différences avec JavaScript sur le mot clé class
  - On doit déclarer les propriétés
  - On peut définir une visibilité pour chaque membre : *public, private, protected*

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



# TypeScript - Classes

- Une classe peut
  - Hériter d'une autre classe (comme en JS)
  - Implémenter une interface
  - Être utilisée comme type

```
interface Writable {  
    write(data: string): void;  
}  
  
class FileLogger implements Writable {  
    write(data: string): Writable {  
        console.log(`Write ${data}`);  
        return this;  
    }  
}
```



# TypeScript - Génériques

- Permet de paramétrer le type de certaines méthodes

```
class Stack<T> {  
    private data: Array<T> = [];  
    push(val: T) {  
        this.data.push(val);  
    }  
    pop(): T {  
        return this.data.pop();  
    }  
    peek(): T {  
        return this.data[this.data.length - 1];  
    }  
}  
  
const strStack = new Stack<string>();  
strStack.push('html');  
strStack.push('body');  
strStack.push('h1');  
console.log(strStack.peek().toUpperCase()); // H1  
console.log(strStack.pop().toUpperCase()); // H1  
console.log(strStack.peek().toUpperCase()); // BODY
```



# TypeScript - Décorateurs

- Permettent l'ajout de fonctionnalités aux classes ou membre d'une classe en annotant plutôt que via du code à l'utilisation
- Norme à l'étude en JavaScript par le TC39  
<https://github.com/tc39/proposal-decorators>
- Supporté de manière expérimentale en TypeScript
- Pour activer leur support il faut éditer le tsconfig.json ou passer une option au compilateur

```
{  
  "compilerOptions": {  
    "target": "es5",  
    "experimentalDecorators": true  
  }  
}
```



# TypeScript - Décorateurs

- Décorateur de classes

```
'use strict';

function Freeze(obj) {
    Object.freeze(obj);
}

@Freeze
class MyMaths {
    static sum(a, b) {
        return Number(a) + Number(b);
    }
}

try {
    MyMaths['subtract'] = function(a, b) {
        return a - b;
    };
}
catch(err) {
    // Cannot add property subtract, object is not extensible
    console.log(err.message);
}
```



# TypeScript - Décorateurs

- Décorateur de propriétés

```
import 'reflect-metadata';

const minLengthMetadataKey = Symbol("minLength");

function MinLength(length: number) {
    return Reflect.metadata(minLengthMetadataKey, length);
}

function validateMinLength(target: any, propertyKey: string): boolean {
    const length = Reflect.getMetadata(minLengthMetadataKey, target, propertyKey);
    return target[propertyKey].length >= length;
}

class Contact {
    @MinLength(7)
    protected firstName;

    constructor(firstName: string) {
        this.firstName = firstName;
    }

    isValid(): boolean {
        return validateMinLength(this, 'firstName');
    }
}

const romain = new Contact('Romain');
console.log(romain.isValid()); // false
```



**formation.tech**

# RxJS

# RxJS - Introduction



- Vers le milieu des années 2000, des bibliothèques comme Bluebird se créent et implémentent le concept de promesse (API Promise)
- L'API Promise devient natif en 2015 avec ES6
- Les promesses permettent de simplifier l'écriture de code de code asynchrone mais ne fonctionnent que pour les callbacks asynchrones appelées une seule fois (setTimeout, requête AJAX, accès à la plupart des bases de données et des systèmes de fichier...)
- Pour répondre aux besoins des callbacks asynchrone appelées plusieurs fois on pourrait remplacer les promesses par des Observables (setInterval, WebSockets, Workers, Evénements Souris / Clavier, Changement d'URL, Bindings...)
- RxJS est une bibliothèque qui implémente le concept tout en ajoutant des fonctionnalités supplémentaires (operators, subjects...)
- L'API Observable est dans le processus pour faire partie de la norme JavaScript

# RxJS - Promise vs Observable



- Création et déclenchement d'une Promise (native ECMAScript)

```
function timeout(delay) {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve(delay);
    }, delay);
  });
}

timeout(1000)
  .then((delay) => console.log(delay + 'ms'));
```

```
$ node timeout.js
1000ms
```



# RxJS - Promise vs Observable

- Création et déclenchement d'une Observable (avec RxJS)

```
function interval(delay) {
  return new Observable((observer) => {
    const intervalId = setInterval(() => {
      observer.next(delay);
    }, delay);

    return () => {
      clearInterval(intervalId);
    };
  });
}

const intervals$ = interval(1000);

const subscription = intervals$
  .subscribe((delay) => console.log(delay + 'ms'));

setTimeout(() => {
  subscription.unsubscribe();
}, 4500);
```

```
$ node interval.js
1000ms
1000ms
1000ms
1000ms
```

# RxJS - Promise vs Observable



- Différences

	Promise (ECMAScript)	Promise (Bluebird)	Observable (RxJS)
Déclenchement du code asynchrone	Au moment de l'appel	Au moment de l'appel	Au moment de la souscription
Nombre d'appels asynchrones	Un	Un	Plusieurs
Etats	succès (then), erreur (catch)	succès (then), erreur (catch)	succès (next), erreur (error), fin (complete)
Annulation	A implémenter	Prévue	Prévue
Combinaisons / Transformations prévues	4 ( <code>Promise.all</code> , <code>.race</code> , <code>.allSettled</code> , <code>.any</code> )	~ 20 méthodes de création ou transformation	~130 méthodes de création ou transformation

# RxJS - Opérateurs



```
import { Component, OnInit, EventEmitter } from '@angular/core';
import { HttpClient } from '@angular/common/http'
import { of } from 'rxjs/observable/of';
import { catchError, debounceTime, distinctUntilChanged, filter, map, switchMap, tap } from 'rxjs/operators';

@Component({
  selector: 'my-app',
  templateUrl: './app.component.html',
})
export class AppComponent implements OnInit {
  public users$;
  public selectedUser;
  public loading;
  public typeahead = new EventEmitter<string>();
  constructor(private httpClient: HttpClient) {}
  ngOnInit() {
    this.users$ = this.typeahead.pipe(
      filter((term) => term.length >= 3),
      distinctUntilChanged(),
      debounceTime(200),
      tap(() => this.loading = true),
      switchMap(
        (term) => this.httpClient.get<any>(`https://api.github.com/search/users?q=${term}`).pipe(
          catchError(() => of({items: []})),
          map(rsp => rsp.items),
          tap(() => this.loading = false),
        )
      );
  }
}
```

# RxJS - Opérateurs



- Dans l'exemple précédent, typeahead est un observable qui fournit dans le temps les valeurs saisies dans un champ

- *filter*

L'observable résultant n'émettra une valeur que si la condition est vérifiée

```
----{h}----{he}----{hel}----{hell}---{hello}-----  
filter((term) => term.length >= 3)  
-----{hel}----{hell}---{hello}-----
```

- *distinctUntilChanged*

L'observable résultant n'émettra une valeur que si elle diffère de la précédente

```
----{hel}--{hel}----{hell}----{hell}---{hello}-----  
distinctUntilChanged()  
----{hel}-----{hell}-----{hello}-----
```

# RxJS - Opérateurs



- *debounceTime*

L'observable résultant n'émettra une valeur que s'il se produit une pause de n millisecondes entre 2 valeurs

```
----{h}---{he}----{hel}-----{hell}---{hello}-----  
debounceTime(400)  
-----{he}-----{hel}-----{hello}--
```

(un tiret vaut 100 ms)

- *tap*

L'observable est identique au précédent, le callback est appelé pour chaque valeur mais n'influe pas sur l'observable (side-effect)

```
----{hel}--{hel}----{hell}-----{hell}---{hello}-----  
tap(() => console.log('Hello'))  
----{hel}--{hel}----{hell}-----{hell}---{hello}-----
```

(affiche également 5 fois 'Hello' dans la console)

# RxJS - Opérateurs



- *map*

L'observable résultant verra ses valeurs transformées par le callback de map

```
----{h}---{he}----{hel}-----{hell}---{hello}-----  
map((v) => v.toUpperCase())  
----{H}---{HE}----{HEL}-----{HELL}---{HELLO}-----
```

- *switchMap*

Combine 2 observables, le second étant créé à partir d'une valeur provenant du premier

```
----{1}-----{2}-{3}---{4}-----  
-----{Res1}-----{Res3}--{Res4}-----{Res2}-  
switchMap((id) => this.http.get(...))  
-----{Res1}-----{Res4}-----
```

(les requêtes 2 et 3 sont annulées car la 3 a commencée avant le retour de la seconde et la 4 avant le retour de la 3)

# RxJS - Allez plus loin



- What's with the Subjects in RxJS 5  
<https://samvloeberghs.be/posts/whats-with-the-subjects-in-rxjs5>
- <https://rxmarbles.com/>
- <https://www.learnrxjs.io/>
- <https://reactive.how/rxjs/explorer>
-



**formation.tech**

# Zone.js



# Zone.js - Introduction

- Angular inclus une bibliothèque développée par Google appeler Zone.js
- Zone.js permet d'intercepter automatiquement des callbacks asynchrone et d'exécuter du code avant ou après
- Angular s'en sert principalement dans 3 contextes :
  - Lancer son algo de détection de changement après une requête AJAX, un changement de route ou toute autre opérations asynchrone
  - Interceptor les erreurs pouvant se produire dans les callbacks asynchrones
  - Lors des tests automatisés, marquer la fin du tests à l'issue de l'exécution du code synchrone et asynchrone
- Il est possible de gagner en performance en supprimant Zone.js et en lançant la détection de changement manuellement
- Installation  
`npm install zone.js`



# Zone.js - Exemples

- Exécuter du code après le dernier callback asynchrone

```
require('zone.js');

const myZone = Zone.current.fork({
  onHasTask(delegate, current, target, hasTaskState) {
    if (!hasTaskState.microTask && !hasTaskState.macroTask) {
      console.log('DONE');
    }
  },
});

myZone.run(() => {
  setTimeout(() => {
    console.log('setTimeout 1');
  }, Math.floor(Math.random() * 1001));

  setTimeout(() => {
    console.log('setTimeout 2');
  }, Math.floor(Math.random() * 1001));
});
```

```
setTimeout 2
setTimeout 1
DONE
```



# Zone.js - Exemples

- Exécuter du code avant ou après chaque callback asynchrone

```
require('zone.js');

const myZone = Zone.current.fork({
  onInvokeTask: (parentZoneDelegate, currentZone, targetZone, task) => {
    console.log('async call');
    return parentZoneDelegate.invokeTask(targetZone, task);
    // angular -> $digest
    // dt.detectChanges();
  }
});

myZone.run(() => {
  setTimeout(() => {
    console.log('setTimeout 500ms');
  }, 500);

  setTimeout(() => {
    console.log('setTimeout 800ms');
  }, 800);
});
```

```
async call
setTimeout 500ms
async call
setTimeout 800ms
```



# Zone.js - Exemples

- Intercepter les erreurs dans les callbacks asynchrones

```
require('zone.js');

const myZone = Zone.current.fork({
  onHandleError: (parentZoneDelegate, currentZone, targetZone, error) => {
    console.log(error.message);
  }
});

myZone.run(() => {
  setTimeout(() => {
    throw new Error('Error in async callback : setTimeout 300ms');
  }, 300);

  setTimeout(() => {
    console.log('setTimeout 500ms');
  }, 500);

  setTimeout(() => {
    throw new Error('Error in async callback : setTimeout 800ms');
  }, 800);
});
```

```
Error in async callback : setTimeout 300ms
setTimeout 500ms
Error in async callback : setTimeout 800ms
```



**formation.tech**

# Angular CLI



# Angular CLI - Introduction

- Angular introduit un programme en ligne de commande permettant d'interagir avec l'application :
  - créer un projet
  - builder
  - lancer le serveur de dev
  - générer du code
  - générer les fichiers de langue
  - ...



# Angular CLI - Introduction

- Installation
  - `npm install -g @angular/cli`
- Documentation
  - <https://cli.angular.io/>
  - <https://github.com/angular/angular-cli/wiki>
  - `ng help`
  - `ng help COMMAND`
  - `ng COMMAND --help`



# Angular CLI - Création d'un projet

- Création d'un projet  
`ng new CHEMIN_VERS_MON_PROJET`
- Autres options
  - `--skip-commit`: ne fait pas de commit initial
  - `--routing`: créer un module pour les routes (Single Page Application)
  - `--prefix`: change le préfixe des composant (par défaut *app*)
  - `--style`: change type de fichier CSS (css par défaut ou *sass*, *scss*, *less*, *stylus*)
  - `--service-worker`: ajouter un service worker pour le mode hors-ligne
- Créer un projet sans installer  
`npx -p @angular/cli ng new CHEMIN_VERS_MON_PROJET`



# Angular CLI - Squelette

- **angular.json**  
Fichier de configuration du programme *ng*, permet de renommer des répertoires, des fichiers
- **e2e**  
Test End to End (qui pilotent le navigateur)
- **src/app**  
Le code source de l'application
- **tsconfig.json**  
Configuration du compilateur TypeScript
- **tslint.json**  
Configuration des conventions de code

```
├── README.md
├── angular.json
└── e2e
    ├── protractor.conf.js
    └── src
        ├── app.e2e-spec.ts
        └── app.po.ts
    └── tsconfig.e2e.json
└── node_modules
└── package.json
└── src
    ├── app
    │   ├── app.component.css
    │   ├── app.component.html
    │   ├── app.component.ts
    │   └── app.module.ts
    ├── assets
    ├── browserslist
    ├── environments
    │   └── environment.prod.ts
    └── favicon.ico
    └── index.html
    └── karma.conf.js
    └── main.ts
    └── polyfills.ts
    └── styles.css
    └── test.ts
    └── tsconfig.app.json
    └── tsconfig.spec.json
    └── tslint.json
└── tsconfig.json
└── tslint.json
```



# Angular CLI - Squelette

- **src/assets**  
Les fichiers statiques non-buildés (images...)
- **src/browserslist**  
Les navigateurs ciblés (pour autoprefixer)
- **src/environments**  
Configuration de l'application
- **src/index.html – src/main.ts**  
Points d'entrées de l'application
- **src/polyfills.ts**  
Chargement des polyfills (core-js, ...)
- **src/style.css**  
CSS global
- **src/karma.conf.js – src/test.ts**  
Configuration des tests

```
├── README.md
├── angular.json
└── e2e
    ├── protractor.conf.js
    └── src
        ├── app.e2e-spec.ts
        └── app.po.ts
    └── tsconfig.e2e.json
└── node_modules
└── package.json
└── src
    ├── app
    │   ├── app.component.css
    │   ├── app.component.html
    │   ├── app.component.ts
    │   └── app.module.ts
    ├── assets
    ├── browserslist
    ├── environments
    │   ├── environment.prod.ts
    │   └── environment.ts
    ├── favicon.ico
    ├── index.html
    ├── karma.conf.js
    ├── main.ts
    ├── polyfills.ts
    ├── styles.css
    ├── test.ts
    └── tsconfig.app.json
    └── tsconfig.spec.json
    └── tslint.json
└── tsconfig.json
└── tslint.json
```



# Angular CLI - Build

- Compiler l'application Angular
  - `ng build`
- Options intéressantes :
  - `--prod` : minifie le code avec Terser et active les options `--aot`, `--environment=prod`, `--extract-css`, `--build-optimizer...`
  - `--environment=NOM` : permet de charger un fichier de configuration particulier (staging, test...)
  - `--vendor-chunk` : pour que le code de node\_modules soit dans un fichier séparé



# Angular CLI - Build

- Gains d'un build avec `--prod`

```
Angular 4 : ng build
Date: 2017-11-02T09:02:41.042Z
Hash: 1d2842c3e0ac46a944f0
Time: 6349ms
chunk {inline} inline.bundle.js, inline.bundle.js.map (inline) 5.83 kB [entry] [rendered]
chunk {main} main.bundle.js, main.bundle.js.map (main) 18.1 kB {vendor} [initial] [rendered]
chunk {polyfills} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 199 kB {inline} [initial] [rendered]
chunk {styles} styles.bundle.js, styles.bundle.js.map (styles) 11.3 kB {inline} [initial] [rendered]
chunk {vendor} vendor.bundle.js, vendor.bundle.js.map (vendor) 1.98 MB [initial] [rendered]
```

```
Angular 5 : ng build
Date: 2017-11-02T09:07:47.401Z
Hash: d1a929eaad03e8e746bb
Time: 4937ms
chunk {inline} inline.bundle.js, inline.bundle.js.map (inline) 5.83 kB [entry] [rendered]
chunk {main} main.bundle.js, main.bundle.js.map (main) 17.9 kB [initial] [rendered]
chunk {polyfills} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 199 kB [initial] [rendered]
chunk {styles} styles.bundle.js, styles.bundle.js.map (styles) 11.3 kB [initial] [rendered]
chunk {vendor} vendor.bundle.js, vendor.bundle.js.map (vendor) 2.29 MB [initial] [rendered]
```

```
Angular 4 : ng build --prod
Date: 2017-11-02T09:03:29.639Z
Hash: cb067f695303856c2315
Time: 6228ms
chunk {0} polyfills.14173651b8ae6311a4b5.bundle.js (polyfills) 61.4 kB {4} [initial] [rendered]
chunk {1} main.f5677287cea9969f6fb6.bundle.js (main) 8.39 kB {3} [initial] [rendered]
chunk {2} styles.d41d8cd98f00b204e980.bundle.css (styles) 0 bytes {4} [initial] [rendered]
chunk {3} vendor.43700a281455e3959c70.bundle.js (vendor) 217 kB [initial] [rendered]
chunk {4} inline.6b5a62abf05dcccfc24d7.bundle.js (inline) 1.45 kB [entry] [rendered]
```

```
Angular 5 : ng build --prod --vendor-chunk
Date: 2017-11-02T09:10:58.444Z
Hash: cf4dd52226e15e33c748
Time: 11487ms
chunk {0} polyfills.ad37cd45a71cb38eee76.bundle.js (polyfills) 61.1 kB [initial] [rendered]
chunk {1} main.2c7fbf970f7125d9617e.bundle.js (main) 7.03 kB [initial] [rendered]
chunk {2} styles.d41d8cd98f00b204e980.bundle.css (styles) 0 bytes [initial] [rendered]
chunk {3} vendor.719fe92af8c44a7e3dac.bundle.js (vendor) 167 kB [initial] [rendered]
chunk {4} inline.220ce59355d1cb2bcb28.bundle.js (inline) 1.45 kB [entry] [rendered]
```



# Angular CLI - Build

## › Differential Loading

```
Angular 8 : ng build
Generating ES5 bundles for differential loading...
ES5 bundle generation complete.

chunk {polyfills-es5} polyfills-es5.js, polyfills-es5.js.map (polyfills-es5) 683 kB [initial]
[rendered]
chunk {styles} styles-es2015.js, styles-es2015.js.map (styles) 9.71 kB [initial] [rendered]
chunk {styles} styles-es5.js, styles-es5.js.map (styles) 11 kB [initial] [rendered]
chunk {runtime} runtime-es2015.js, runtime-es2015.js.map (runtime) 6.12 kB [entry] [rendered]
chunk {runtime} runtime-es5.js, runtime-es5.js.map (runtime) 6.16 kB [entry] [rendered]
chunk {main} main-es2015.js, main-es2015.js.map (main) 20 kB [initial] [rendered]
chunk {main} main-es5.js, main-es5.js.map (main) 23.4 kB [initial] [rendered]
chunk {vendor} vendor-es2015.js, vendor-es2015.js.map (vendor) 3.48 MB [initial] [rendered]
chunk {vendor} vendor-es5.js, vendor-es5.js.map (vendor) 4.2 MB [initial] [rendered]
chunk {polyfills} polyfills-es2015.js, polyfills-es2015.js.map (polyfills) 264 kB [initial] [rendered]
Date: 2019-12-24T13:16:33.881Z - Hash: ed6bb442e1097c9cd39e - Time: 6430ms
```

```
Angular 8 : ng build --prod --vendor-chunk
```

*Generating ES5 bundles for differential loading...*  
*ES5 bundle generation complete.*

```
chunk {3} polyfills-es5.6696c533341b95a3d617.js (polyfills-es5) 123 kB [initial] [rendered]
chunk {2} polyfills-es2015.2987770fde9daa1d8a2e.js (polyfills) 36.4 kB [initial] [rendered]
chunk {0} runtime-es2015.edb2fcf2778e7bf1d426.js (runtime) 1.45 kB [entry] [rendered]
chunk {0} runtime-es5.edb2fcf2778e7bf1d426.js (runtime) 1.45 kB [entry] [rendered]
chunk {1} main-es2015.a0a4227c812d9c1eacbc.js (main) 3.8 kB [initial] [rendered]
chunk {1} main-es5.a0a4227c812d9c1eacbc.js (main) 3.97 kB [initial] [rendered]
chunk {5} vendor-es2015.4109580e16948d8caeb4.js (vendor) 144 kB [initial] [rendered]
chunk {5} vendor-es5.4109580e16948d8caeb4.js (vendor) 172 kB [initial] [rendered]
chunk {4} styles.3ff695c00d717f2d2a11.css (styles) 0 bytes [initial] [rendered]
Date: 2019-12-24T13:15:28.783Z - Hash: e42078e85114e028a1c8 - Time: 11200ms
```

# Angular CLI - Serveur de développement



- Lancer le serveur de dev
  - `ng serve`
- Options intéressantes :
  - `--port` : changer le port
  - `--target=production` : sert les fichiers dans la config de prod



# Angular CLI - Générateurs

- Générateurs
  - Angular CLI contient un certains nombre de générateurs : application, class, component, directive, enum, guard, interface, module, pipe, service, universal, appShell
- Dry run
  - Chaque générateur peut se lancer avec l'option `--dry-run` ou `-d` qui va afficher le résultat de la commande sans rien créer, sachant qu'il n'y a pas de retour automatique possible une fois les fichiers créés.
- Afficher la doc d'un générateur
  - `ng help generate NOM_DU_GENERATEUR`



# Angular CLI - Générateurs

- Générer un module
  - `ng generate module CHEMIN_DEPUIS_APP`
  - `ng g m CHEMIN_DEPUIS_APP`
- Autres options
  - `--routing`: génère un 2e module pour les routes
  - `--flat`: ne créé pas de répertoire



# Angular CLI - Générateurs

- Générer un composant
  - `ng generate component CHEMIN_DEPUIS_APP`
  - `ng g c CHEMIN_DEPUIS_APP`
- Autres options
  - `--flat` : ne crée pas de répertoire
  - `--export` : ajoute une entrée dans les exports du module



# Angular CLI - Tests & lint

- Lancer les tests Karma + Jasmine
  - `ng test`
- Lancer les tests Protractor
  - `ng e2e`
- Vérifier les conventions de code
  - `ng lint`
  - `ng lint --fix --type-check`



**formation.tech**

# Composants

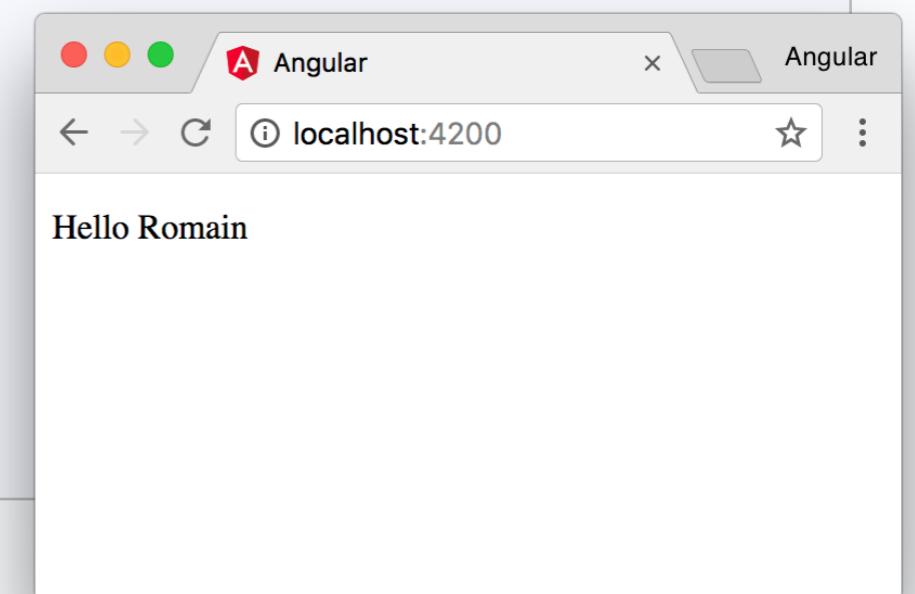


# Composants - Introduction

- 2 parties
  - code TypeScript
  - template
- Compilation
  - Les 2 sont compilés dans un code optimisé pour la VM JavaScript
  - Le template peut être compilé en JIT (par le browser) ou en AOT (au moment du build)

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-hello',
  template: '<p>Hello {{name}}</p>',
})
export class HelloComponent {
  public name = 'Romain';
}
```





# Composants - Lifecycle Hooks

- Les composants peuvent implémenter les interfaces et leurs méthodes suivantes seront appelées automatiquement :
  - OnChanges / ngOnChanges() : lorsque qu'un changement se produit au niveau d'un input binding.
  - OnInit / ngOnInit() : une fois que le composant reçoit ses propriétés @Input et affiche les premiers input bindings.
  - OnDestroy / ngOnDestroy() : juste avant la destruction du component/directive. Il faut s'y désabonner des Observable ou événement pour éviter les fuites mémoires.
  - DoCheck / ngDoCheck() à chaque lancement de la détection de changement, permet d'identifier des changements que ngOnChanges ne peut détecter.

# Composants - Lifecycle Hooks



- `ngAfterContentInit()`
  - Respond after Angular projects external content into the component's view / the view that a directive is in.
  - Called once after the first `ngDoCheck()`.
- `ngAfterContentChecked()`
  - Respond after Angular checks the content projected into the directive/component.
  - Called after the `ngAfterContentInit()` and every subsequent `ngDoCheck()`.
- `AfterViewInit / ngAfterViewInit()`
  - Respond after Angular initializes the component's views and child views / the view that a directive is in.
  - Called once after the first `ngAfterContentChecked()`.
- `AfterViewChecked / ngAfterViewChecked()`
  - Respond after Angular checks the component's views and child views / the view that a directive is in.
  - Called after the `ngAfterViewInit` and every subsequent `ngAfterContentChecked()`.



# Composants - Lifecycle Hooks

- Exemple

```
import { Component, OnDestroy, OnInit } from '@angular/core';

@Component({
  selector: 'hello-lifecycle',
  template: `
    {{ now | date:'HH:mm:ss' }}
  `,
})
export class LifecycleComponent implements OnInit, OnDestroy {

  public now = new Date();
  private intervalId: number;

  ngOnInit() {
    this.intervalId = setInterval(() => {
      this.now = new Date();
    }, 1000)
  }

  ngOnDestroy() {
    clearInterval(this.intervalId);
  }

}
```



**formation.tech**

# Templates



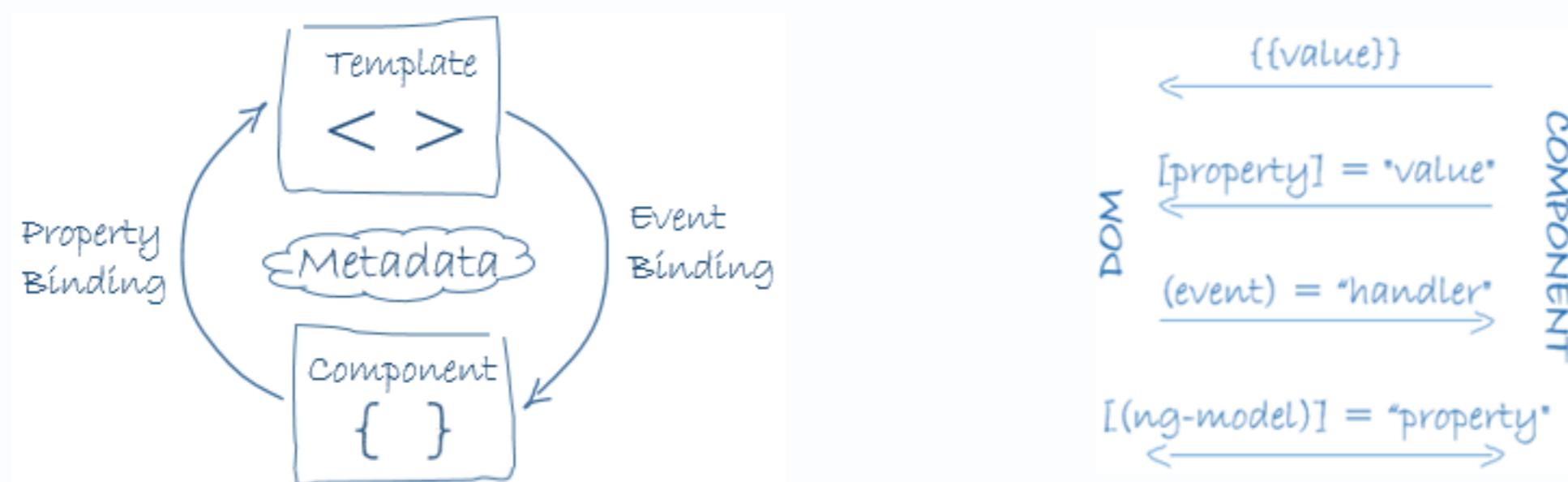
# Templates - Introduction

- Templates
  - Comme dans AngularJS, on décrit l'interface de manière déclarative dans des templates
  - Chaque template est compilé par le compilateur d'Angular, soit en amont (mode AOT pour Ahead Of Time Compilation), soit dans le browser (mode JIT pour Just In Time Compilation)
  - Les templates sont ainsi transformé en du code optimisé pour la VM/Moteur JavaScript



# Templates - Data binding

- Data binding
  - Sans data binding ce serait au développeur de maintenir les changements à opérer sur le DOM à chaque événement
  - Dans jQuery par exemple, cliquer sur un bouton peut avoir pour conséquence de rafraîchir une balise, de lancer un indicateur de chargement...
  - Avec Angular le développeur décrit l'état du DOM en fonction de propriétés qui constituent le Modèle, ainsi un événement n'a plus qu'





# Templates - Property Binding

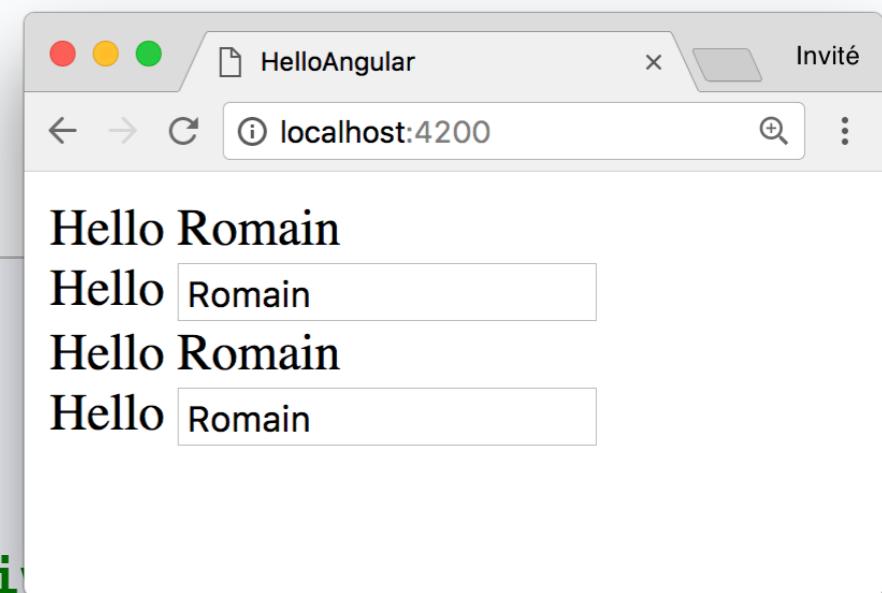
- 2 syntaxes

Pour synchroniser le DOM avec le modèle (les propriétés publiques du composant dans Angular)

- bind-nomDeLaPropDuDOM="propDuComposant"
- [nomDeLaPropDuDOM]="propDuComposant"

```
import { Component } from '@angular/core';

@Component({
  selector: 'hello-property-binding',
  template: `
    <div>Hello <span bind-textContent="prenom"></span></div>
    <div>Hello <input bind-value="prenom"></div>
    <div>Hello <span [textContent]="prenom"></span></div>
    <div>Hello <input [value]="prenom"></div>
  `,
})
export class PropertyBindingComponent {
  public prenom = 'Romain';
}
```



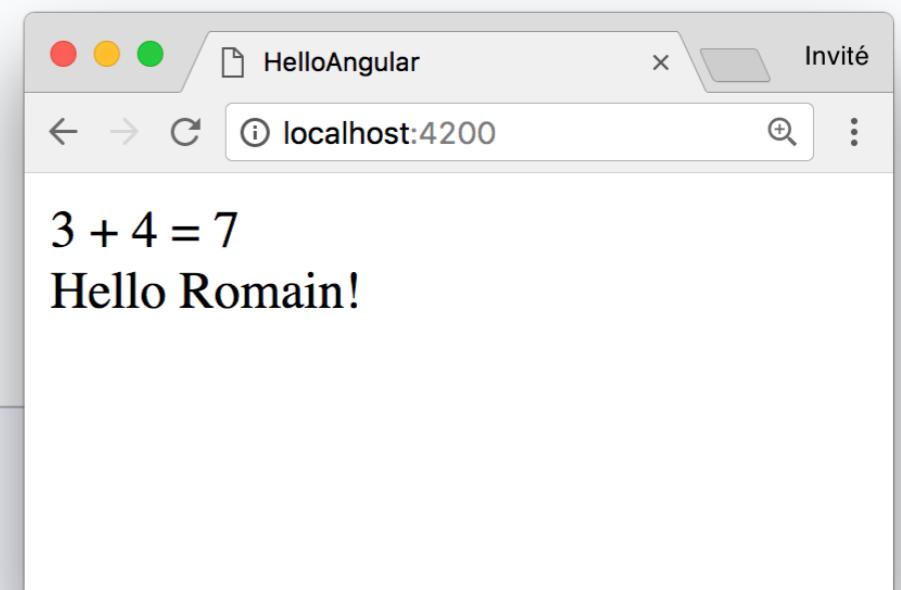


# Templates - Expressions

- Dans un property binding il est possible d'utiliser des noms de propriétés ou des expressions, sauf les expressions ayant des effets de bords :
  - affectations (`=, +=, -=, ...`)
  - `new`
  - expressions chainées avec ; ou ,
  - incrementation et décrémentation (`++` et `--`)

```
import { Component } from '@angular/core';

@Component({
  selector: 'hello-prenom',
  template: `
    <div>3 + 4 = <span [textContent]="'3 + 4'"></span></div>
    <div>Hello <span [textContent]="'prenom + ' !'"></span></div>
  `,
})
export class PrenomComponent {
  public prenom = 'Romain';
}
```





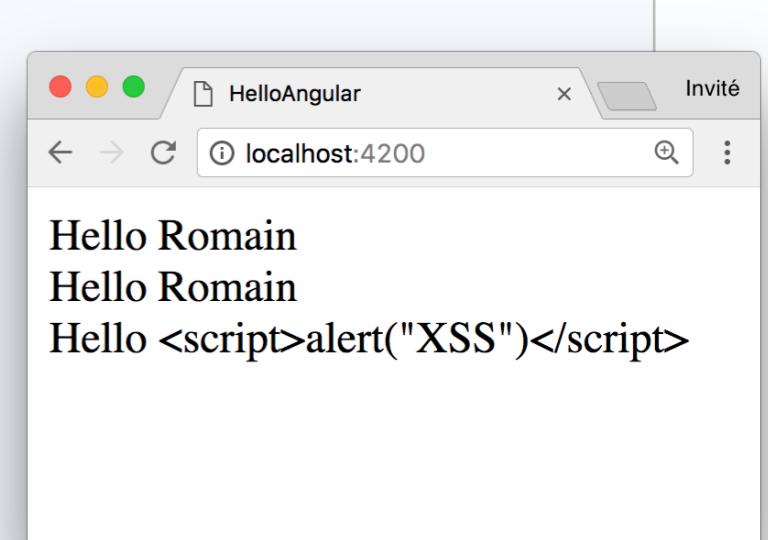
# Templates - Interpolation

- Interpolation

- Plutôt que bind-innerHTML sur une balise span, on peut utiliser la syntaxe aux doubles accolades {{ }}
- A privilégier car cette syntaxe échappe les entrées, évitant ainsi que des balises contenues dans les entrées se retrouvent dans le DOM (faille XSS)

```
import { Component } from '@angular/core';

@Component({
  selector: 'hello-interpolation',
  template: `
<div>Hello <span [innerHTML]="prenom"></span></div>
<div>Hello {{prenom}}</div>
<div>Hello {{xssAttack}}</div>
  `,
})
export class InterpolationComponent {
  public prenom = 'Romain';
  public xssAttack = '<script>alert("XSS")</script>';
}
```





# Templates - Event Binding

- 2 syntaxes

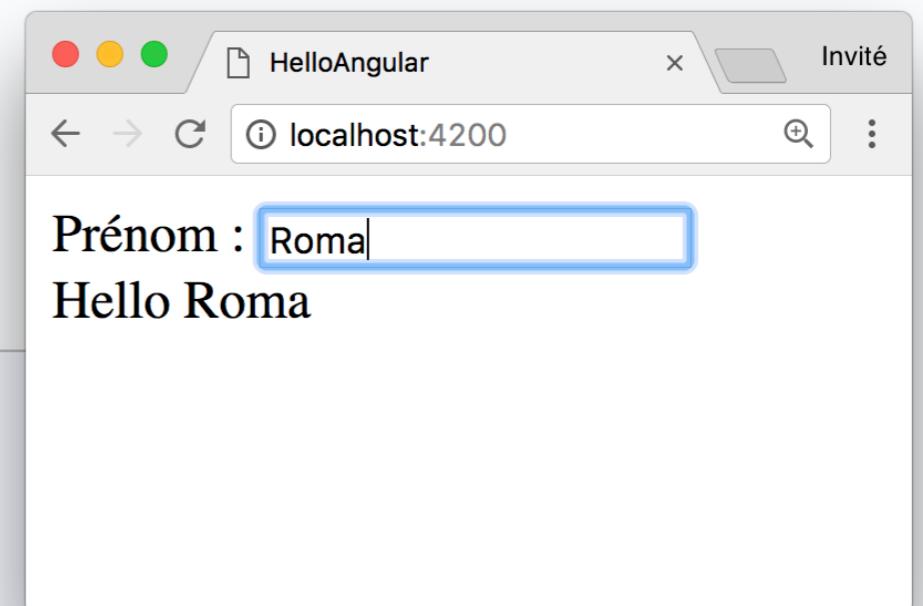
Pour synchroniser le DOM avec le modèle (les propriétés publiques du composant dans Angular), on utilise des événements

- `on-nomEvent="expression"`
- `(nomEvent)="expression"`

```
import { Component } from '@angular/core';

@Component({
  selector: 'hello-event-binding',
  template: `
    <div>Prénom : <input on-input="updatePrenom($event)"></div>
    <div>Prénom : <input (input)="updatePrenom($event)"></div>
    <div>Prénom : <input (input)="prenom = $event.target.value"></div>
    <div>Hello {{prenom}}</div>
  `,
})
export class EventBindingComponent {
  public prenom = '';

  public updatePrenom(e) {
    this.prenom = e.target.value;
  }
}
```





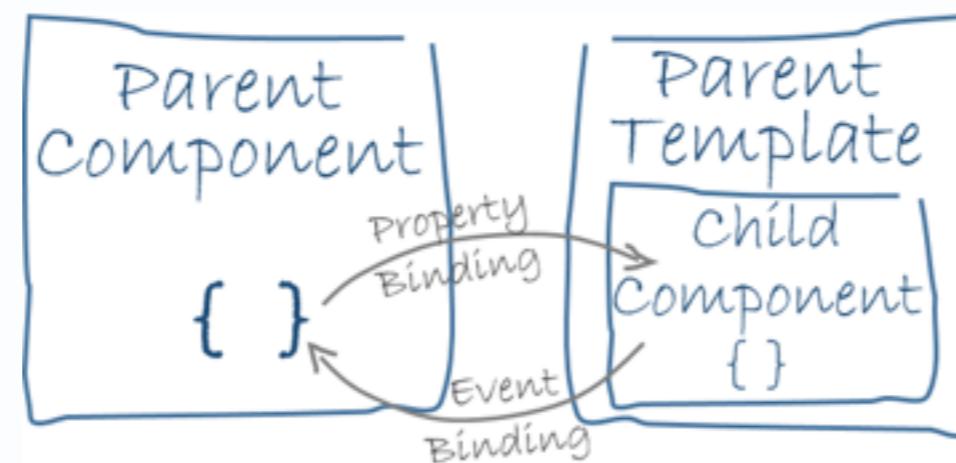
# Templates - Two way data-binding

- AngularJS proposait des bindings dans les 2 sens (two way data-binding )
- Angular propose une syntaxe similaire, mais en réalité le binding se fait toujours en 2 temps, d'abord les events bindings, puis les property bindings
- Pour utiliser les 2 sur une ligne il faut :
  - Avoir un property binding
  - Avoir un event binding du même nom suffixé par change et qui affecte \$event à la variable lié au property binding
- Exemple
  - `[ngModel]="prenom" (ngModelChange)="prenom = $event"`
  - `[(ngModel)]="prenom"`
- Pour moyen mnémotechnique : `[( )]` = BANANA IN A BOX



# Templates - Communication inter-composant

- Pour communiquer entre un composant parent et un composant enfant (imbriqués l'un dans l'autre) on utilise des bindings
- Pour passer des valeurs on utilise des property bindings, la propriété du composant doit alors utiliser le décorateur @Input
- Pour remonter des valeurs au parent on utilise des event bindings, la propriété du composant doit alors utiliser le décorateur @Output et doit être de type EventEmitter





# Templates - Communication inter-composant

- Exemple extrait de ng-select : <https://github.com/ng-select/ng-select>

```
import { Component, EventEmitter, Input, Output } from '@angular/core';

@Component({
  selector: 'ng-select'
})
export class NgSelectComponent {

  @Input() items: any[] = [];

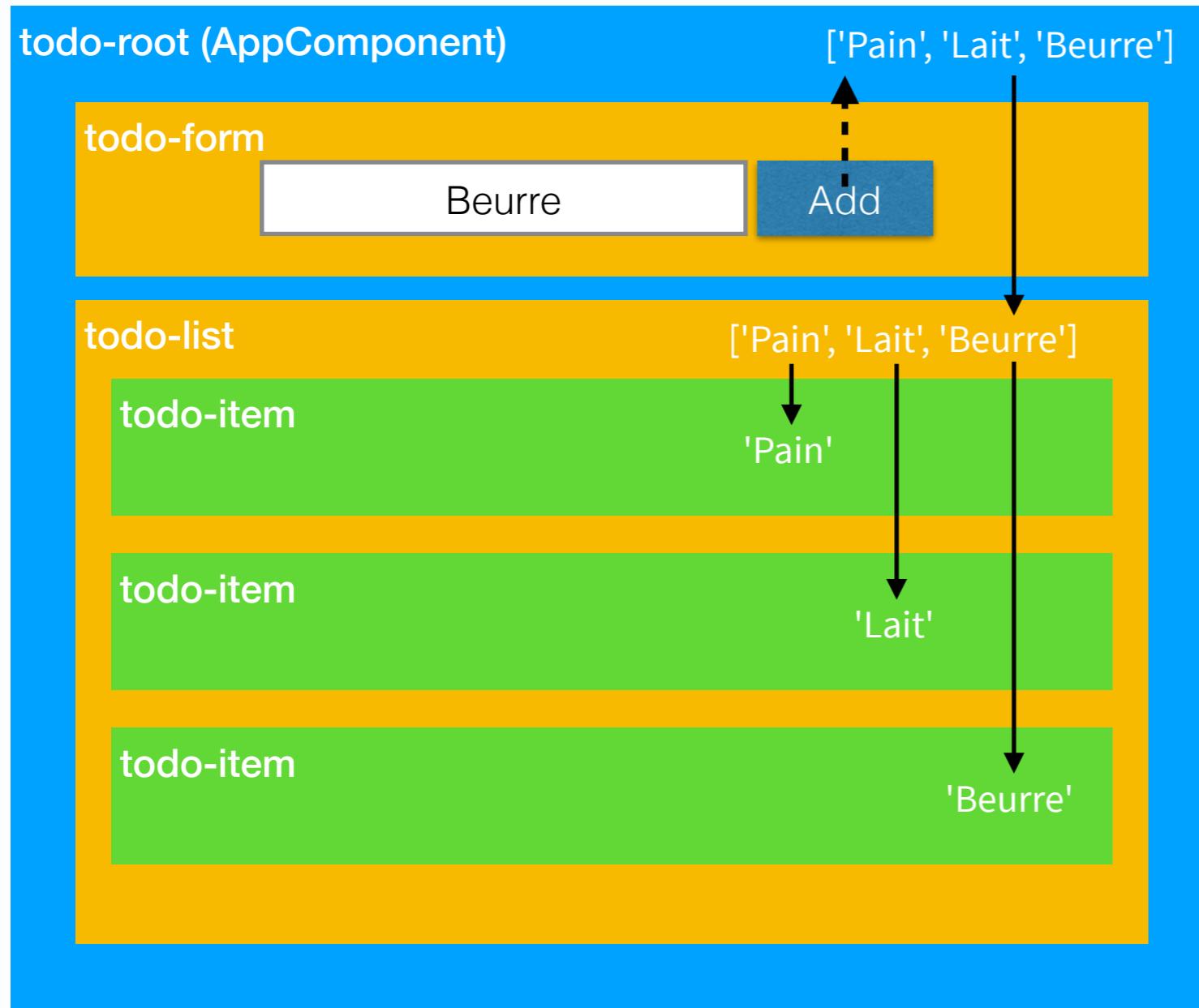
  @Output('add') addEvent = new EventEmitter();

  select(item) {
    // ...
    this.addEvent.emit(item.value);
    // ...
  }
}
```

```
<ng-select [items]="chains" (add)="addChain($event)"></ng-select>
```



# Template - Exercice



- Créer un nouveau projet todolist avec prefix todo et style scss
- Créer 3 composants : Form, List, Item
- Créer un tableau dans App
- Passer le tableau à List via un property binding
- Passer chaque élément du tableau à Item via un property binding
- Au submit du form du composant Form, transmettre la valeur saisie au parent via un Event Binding (ajouter au tableau dans App)



**formation.tech**

# Modules

# Modules - Introduction



- 2 notions de modules
  - NgModule (class décorée avec @NgModule)
  - Module ES6 (import / export de fichiers)
- Jusqu'à la RC d'Angular 2, la notion de NgModule n'existe pas  
Voir TodoMVC : <https://github.com/tastejs/todomvc/tree/gh-pages/examples/angular2>
- Intérêt d'avoir des NgModules :
  - Pouvoir importer un ensemble de composants / directives / pipes...
  - Pour configurer la portée d'un service
  - Permettre de charger des blocs de code par lazy-loading (après le chargement initial)
  - ...

# Modules - Principaux Modules



- Principaux Modules
  - AppModule : le module racine
  - CommonModule : le module qui inclus toutes les directives Angular de base comme *NgIf*, *NgForOf*, mais aussi les pipes comme *DatePipe*, *AsyncPipe*...
  - BrowserModule : exporte *CommonModule* et contient les services permettant le rendu DOM, la gestion des erreurs, la modification des balises *title* ou *meta*...
  - FormsModule : le module qui permet la validation des formulaires, la déclaration de la directive *ngModel*...
  - HttpClientModule : contient les composants pour les requêtes HTTP
  - RouterModule : permet de manipuler des routes (associer des composants à des URL)
- Open-Source

La première chose à faire après l'installation d'une bibliothèque *Angular* via *npm* sera d'importer un module



# Modules - Déclaration

- Déclaration
  - Pour qu'un composant, directive ou pipe existe dans l'application il faut le déclarer dans un module
  - Ne jamais déclarer 2 fois la même classe dans 2 modules différents

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { HelloComponent } from './hello/hello.component';

@NgModule({
  declarations: [
    AppComponent,
    HelloComponent,
  ],
  imports: [
    BrowserModule
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```



# Modules - Erreur courante

- Erreur courante

Si un composant, directive ou pipe n'est pas déclaré dans un module, ou bien que le module dans lequel il est déclaré n'est pas importé par le module qui l'utilise

*Uncaught Error: Template parse errors:*

*'app-title' is not a known element:*

*1. If 'app-title' is an Angular component, then verify that it is part of this NgModule.*

*2. If 'app-title' is a Web Component then add 'CUSTOM\_ELEMENTS\_SCHEMA' to the '@NgModule.schemas' of this component to suppress this message.*

# Modules - Bonnes pratiques



- › Bonnes pratiques

Extrait du Style Guide Angular :

<https://angular.io/guide/styleguide#application-structure-and-ngmodules>

- Créer un dossier core global

Contiendra la déclaration de tous les services mono-instanciés (singleton) et composants, pipes ou directives utilisés uniquement dans le module racine (AppModule)

- Créer un module SharedModule global

Contiendra la déclaration de tous les services multi-instanciés et composants, pipes ou directives utilisés dans différents modules

# Modules - Exports



- Utilisé une déclaration dans un autre module
  - Il est fréquent de vouloir utiliser une déclaration (Composant, Pipe, Directive) dans un autre module dans lequel il est déclaré (ex : ng-select déclaré dans NgSelectModule)
  - Dans ce cas il faut qu'il soit déclaré puis exporté dans un NgModule
  - Ce NgModule devra ensuite être importé par chaque NgModule dans lesquels sera utilisé cette déclaration

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { SelectComponent } from './select/select.component';

@NgModule({
  declarations: [
    SelectComponent,
  ],
  imports: [
    CommonModule
  ],
  exports: [
    SelectComponent,
  ]
})
export class SharedModule { }
```

# Modules - Imports



- Pour importer une déclaration exportée, on importe le module contenant l'export
- La déclaration devient alors accessible dans n'importe quel composant du module
- Dans notre exemple AppComponent a maintenant accès à SelectComponent dans son template

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { SharedModule } from './shared/shared.module';

@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
    SharedModule,
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```



# Modules - Exports de modules

- Il est courant d'exporter un module pour factoriser les imports
- Exemple : si SharedModule exports CommonModule et FormsModule, je n'ai plus qu'à importer SharedModule dans ContactsModule pour pouvoir utiliser NgIf et NgModel

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { SelectComponent } from './select/select.component';

@NgModule({
  declarations: [
    SelectComponent,
  ],
  // en important Common, on peut utiliser NgIf dans SelectComponent
  imports: [
    CommonModule,
  ],
  // si on importe Shared on a accès à SelectComponent mais aussi NgIf, NgModel...
  exports: [
    CommonModule,
    FormsModule,
    SelectComponent,
  ],
})
export class SharedModule { }
```

# Modules - Exports et services



- En important un module, on importe ses déclarations mais également ses services (dont on a décrit l'instanciation avec les providers)
- Le fait d'importer 2 fois un service provoque la réinstanciation du service, c'est à dire qu'il y aura 2 instances dans l'application
- Or il est fréquent de vouloir une seule instance, il faut alors importer le module dans  *AppModule*
- Si le module contient également des déclarations il est courant de retrouver 2 méthodes *forRoot*, *forChild*

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

const routes: Routes = [];

@NgModule({
  // forRoot contient le provider de Router, ActivatedRoute...
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

# Modules - Exports et services



- Exemple avec RouterModule
  - On peut importer RouterModule.forRoot dans AppModule et avoir ainsi une instance unique de Router dans toute l'application
  - On peut importer RouterModule.forChild pour enregistrer de nouvelles routes sans réimporter le service Router
  - On peut importer RouterModule pour importer les directives comme RouterLink ou RouterOutlet

```
@NgModule({
  declarations: ROUTER_DIRECTIVES,
  exports: ROUTER_DIRECTIVES,
})
export class RouterModule {
  static forRoot(routes: Routes, config?: ExtraOptions) {
    return {
      ngModule: RouterModule,
      providers: [ROUTER_PROVIDERS, provideRoutes(routes)],
    };
  }
  static forChild(routes: Routes) {
    return {ngModule: RouterModule, providers: [provideRoutes(routes)]};
  }
}
```



**formation.tech**

# Pipes

# Pipes - Introduction



- Pour pouvoir formater proprement des valeurs
- Dans AngularJS on parlait de filtres
- Les Pipes sont définis dans CommonModule (il faut donc que le composant qui les utilise soit défini dans un Module qui importe CommonModule ou BrowserModule)
- Utilisation

```
{{ now | date:'HH:mm:ss' }}
```

## common

 AsyncPipe	 CurrencyPipe
 DatePipe	 DecimalPipe
 DeprecatedCurrencyPipe	 DeprecatedDatePipe
 DeprecatedDecimalPipe	 DeprecatedPercentPipe
 JsonPipe	 KeyValuePipe
 LowerCasePipe	 PercentPipe
 SlicePipe	 TitleCasePipe
 UpperCasePipe	

# Pipes - Crédit d'un Pipe



- Les Pipes doivent être déclarés dans un module
- Eventuellement exportés pour pouvoir être utilisés ailleurs
- Implémentent *PipeTransform*
- Le premier paramètre de *transform* est la valeur à transformer
- Il est possible de passer des paramètres supplémentaires (ex : format de Date)

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'duration',
})
export class DurationPipe implements PipeTransform {
  transform(seconds: number, args?: any): any {
    const h = String(Math.floor(seconds / (60 * 60))).padStart(2, '0');
    const m = String(Math.floor((seconds % (60 * 60)) / 60)).padStart(2, '0');
    const s = String(seconds % 60).padStart(2, '0');
    return `${h}:${m}:${s}`;
  }
}
```



**formation.tech**

# Services

# Services - Introduction



- Les Services sont des objets utilitaires associés à des concepts informatiques (Client HTTP, Conversion Euro Dollars, Navigation vers un autre composant routé...), par opposition aux Value Objects qui représente des valeurs (String, Date, Model...)
- Angular contient une fonction appelée Injecteur qui permet de retrouver un service facilement sans avoir à le créer directement
- Il faut alors expliquer au framework comment doit se créer le service lorsqu'il est demandé dans un certain contexte
- L'injecteur Angular se base sur le typage statique de TypeScript pour déterminer l'objet à injecter



# Services - Providers

- Un service doit être décoré avec `@Injectable`

```
import { Injectable } from '@angular/core';

@Injectable()
export class UserService {
  users = [
    {
      id: 1,
      name: 'Romain',
      city: 'Paris',
    },
    {
      id: 2,
      name: 'Steven',
      city: 'Besançon',
    }
  ];
}
```

# Services - Providers



- Pour indiquer à Angular comment créer le service on utilise un *provider* au niveau d'un *NgModule*
- *useClass* permet d'indiquer à Angular qu'il saura résoudre toutes les dépendances de ce service (dans ce cas on peut passer directement le nom de la classe plutôt qu'un provider)

```
import { NgModule } from '@angular/core';
import { SharedModule } from './shared/shared.module';
import { UsersComponent } from './users/users.component';
import { UserService } from './user.service';

@NgModule({
  declarations: [
    UsersComponent,
  ],
  imports: [
    SharedModule,
  ],
  providers: [
    { provide: UserService, useClass: UserService },
    // équivalent à :
    UserService
  ],
})
export class UsersModule { }
```

# Services - Providers



- Pour récupérer le service, il suffit de le demander dans le constructeur d'une classe
- Rappelons nous qu'en TypeScript, utiliser public, protected ou private devant un paramètre du constructeur permet de l'affecter directement à la propriété du même nom

```
import { Component, OnInit } from '@angular/core';
import { User } from '../user.model';
import { Observable } from 'rxjs';
import { UserService } from '../user.service';

@Component({
  selector: 'app-users-list',
  templateUrl: './users-list.component.html',
  styleUrls: ['./users-list.component.scss']
})
export class UsersListComponent implements OnInit {
  users$: Observable<User[]>;

  constructor(protected userService: UserService) { }

  ngOnInit() {
    this.users$ = this.userService.getAll$();
  }
}
```



# Services - Providers

- Un service peut lui même recevoir des dépendances

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { User } from './user.model';

@Injectable()
export class UserService {

  constructor(protected httpClient: HttpClient) { }

  getAll$(): Observable<User[]> {
    return this.httpClient.get<User[]>('/users');
  }
}
```



# Services - Providers

- Les dépendances peuvent cependant ne pas être résolue automatiquement (ici url)

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { User } from './user.model';

@Injectable()
export class UserService {

  constructor(protected httpClient: HttpClient, protected url: string) { }

  getAll$(): Observable<User[]> {
    return this.httpClient.get<User[]>(this.url);
  }
}
```



# Services - Singleton

- Dans ce cas on pourra utiliser une fabrique avec *useFactory*
- La fabrique n'étant pas décorée avec *@Injectable*, il faudra lui donner ses Token d'Injection avec *deps*

```
export function userServiceFactory(httpClient: HttpClient) {
  return new UserService(httpClient, '/users');
}

@NgModule({
  declarations: [
    UsersComponent,
  ],
  imports: [
    SharedModule,
  ],
  providers: [
    { provide: UserService, useFactory: userServiceFactory, deps:
[HttpClient] },
  ],
})
export class UsersModule { }
```



# Services - Singleton

- Pour garantir qu'un service n'ait qu'une seule instance il faut le déclarer dans AppModule ou dans un module qui ne sera inclus que par AppModule
- Depuis Angular 6 il est possible de déclarer un service directement dans le décorateur @Injectable, 'root' signifiant AppModule

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { User } from './user.model';

@Injectable({
  providedIn: 'root'
})
export class UserService {

  constructor(protected httpClient: HttpClient) { }

  getAll$(): Observable<User[]> {
    return this.httpClient.get<User[]>('/users');
  }
}
```



**formation.tech**

# Routeur



# Routeur - Introduction

- Le routeur permet de créer une Single Page Application (SPA) c'est à dire où le passage d'une page à une autre se fera en JavaScript
- Le navigateur n'a pas à rafraîchir tout le site, mais seulement le composant routé, le passage d'une page à une autre pourra même être animé
- Il manquait au routeur d'AngularJS un certain nombre de fonctionnalités qui ont été introduites par ui-router
- Le routeur officiel d'Angular s'est inspiré dès sa conception des routeurs les plus modernes

# Routeur - Principales fonctionnalités



- Principales fonctionnalités :
  - routage de composant avec ou sans paramètre
  - Resolvers pour attendre les données avant le changement de page
  - Guards pour vérifier les droits avant l'accès à une page
  - Services pour intéragir avec les données de la route et les
  - Evénements pour écouter globalement le changement de route
  - plusieurs routes par URL en nommant les conteneurs de composants
  - plusieurs routes par URL en imbriquant les routes
  - lazy loading permettant le chargement du code d'un module uniquement lors de l'accès à l'une de ses pages



# Routeur - Mise en place

- › Par convention on charge les routes dans des modules associés
- › Pour créer un projet avec le routeur actif (va créer un app-routing.module) :  
`ng new --routing NOM_DU_PROJET`
- › Pour créer un feature module avec des routes  
`ng generate module --routing NOM_DU_MODULE`
- › Il faut parfois redémarrer le live-server pour que les routes d'un module soit prisent en compte



# Routeur - AppRoutingModule

- On remarque que les routes charges via la méthodes *forRoot* de *RouterModule* qui permet que des services comme *Router* soient disponible sous forme de Singleton
- Les URLs/paths ne commencent pas par des slashes
- **\*\*** est une wildcard, utilisée ici pour les erreurs 404

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './core/home/home.component';
import { NotFoundComponent } from './core/not-found/not-found.component';

const routes: Routes = [
  {
    path: '',
    component: HomeComponent,
  },
  {
    path: '**',
    component: NotFoundComponent,
  }];
  
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```



# Routeur - Features Modules

- Il n'y a que *AppRoutingModule* qui appelle *forRoot*, les autres features modules doivent appeler la méthode *forChild* (qui ne refournit pas les services)
- RouterModule est exporté et donc importé en même temps que le module de routage, ce qui rend ses directives accessibles dans les composants déclarés dans AppModule

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { UsersComponent } from './users/users.component';

const routes: Routes = [
  path: 'users',
  component: UsersComponent,
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class UsersRoutingModule { }
```

# Routeur - Import de routes



- L'ordre d'import est important, il est faudra importer AppRoutingModule en dernier s'il contient une wildcard
- Idem par la suite avec les routes avec paramètres, il est important de les créer en second pour éviter les conflits

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { SharedModule } from './shared/shared.module';
import { UsersModule } from './users/users.module';

@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
    UsersModule,
    AppRoutingModule,
    SharedModule,
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```



# Routeur - Directives

- RouterOutlet

Permet d'indiquer à Angular où doit être inséré le composant routé

```
<app-top-bar></app-top-bar>
<div class="container-fluid pt-3">
  <router-outlet></router-outlet>
</div>
```

- On peut avoir plusieurs *router-outlet* par page, mais il faudra alors ajouter la clé *outlet* à la définition de la route

```
<app-top-bar></app-top-bar>
<div class="container-fluid pt-3">
  <div class="row">
    <div class="col-lg-3">
      <router-outlet name="left"></router-outlet>
    </div>
    <div class="col-lg">
      <router-outlet name="right"></router-outlet>
    </div>
  </div>
</div>
```

```
{
  path: '',
  component: HomeComponent,
  outlet: 'left'
},
```

# Routeur - Directives



- RouterLink

Permet de créer un lien, on peut soit lui donner une URL, soit un tableau de paths (préférable si les routes sont imbriquées)

```
<a class="nav-link" routerLink="/">Home</a>
<a class="nav-link" [routerLink]=["['users']"]>Users</a>
```

- RouterLinkActive

Ajoute la classe en paramètre (ici active) à l'élément commençant par le lien actif *routerLinkActiveOptions* permet de mettre l'option *exact* à *true* et force le lien complet

```
<div class="navbar-nav nav">
  <div class="nav-item"
    routerLinkActive="active" [routerLinkActiveOptions]="{{exact: true}}"
      <a class="nav-link" routerLink="/">Home</a>
    </div>
    <div class="nav-item" routerLinkActive="active">
      <a class="nav-link" [routerLink]=["['users']"]>Users</a>
    </div>
  </div>
```



# Routeur - Paramètres

- Pour définir des routes avec paramètres on définit le paramètre avec *:nom\_du\_param*
- Le paramètre sera ensuite accessible dans le composant routé avec le service *ActivatedRoute*

```
const routes: Routes = [ {  
  path: 'users/:id',  
  component: UserShowComponent,  
}];
```

# Routeur - ActivatedRoute



- Le service *ActivatedRoute* permet d'accéder aux paramètres sous forme d'*Observable* via sa clé *paramMap* (la clé *params* pourrait être dépréciée à l'avenir)
- On peut également obtenir un *snapshot* au lieu d'un *Observable*, mais le passage d'une URL à une autre URL activant le même composant ne va pas réinstancier celui-ci, on perdrait alors les prochains changements de paramètre
- D'autres clés sont utiles comme *queryParamMap* ou *data* (des données personnalisées comme le titre de la page définies au niveau de la route)

```
export class UserShowComponent implements OnInit {  
  
  constructor(  
    protected activatedRoute: ActivatedRoute,  
  ) { }  
  
  ngOnInit() {  
    this.activatedRoute.paramMap.subscribe((params) => {  
      console.log(params.get('id'));  
    });  
  }  
}
```

# Routeur - Router Service



- Le service *Router* permet de naviguer d'une route à une autre en TypeScript
- La clé *routerState* permet de naviguer dans l'arbre des routes activées

```
export class UserAddComponent {  
  
  user = new User();  
  
  constructor(  
    protected userService: UserService,  
    protected router: Router,  
  ) {}  
  
  createUser() {  
    this.userService.create$(this.user).subscribe((user) => {  
      this.router.navigate(['users', user.id]);  
    });  
  }  
}
```

# Routeur - Events



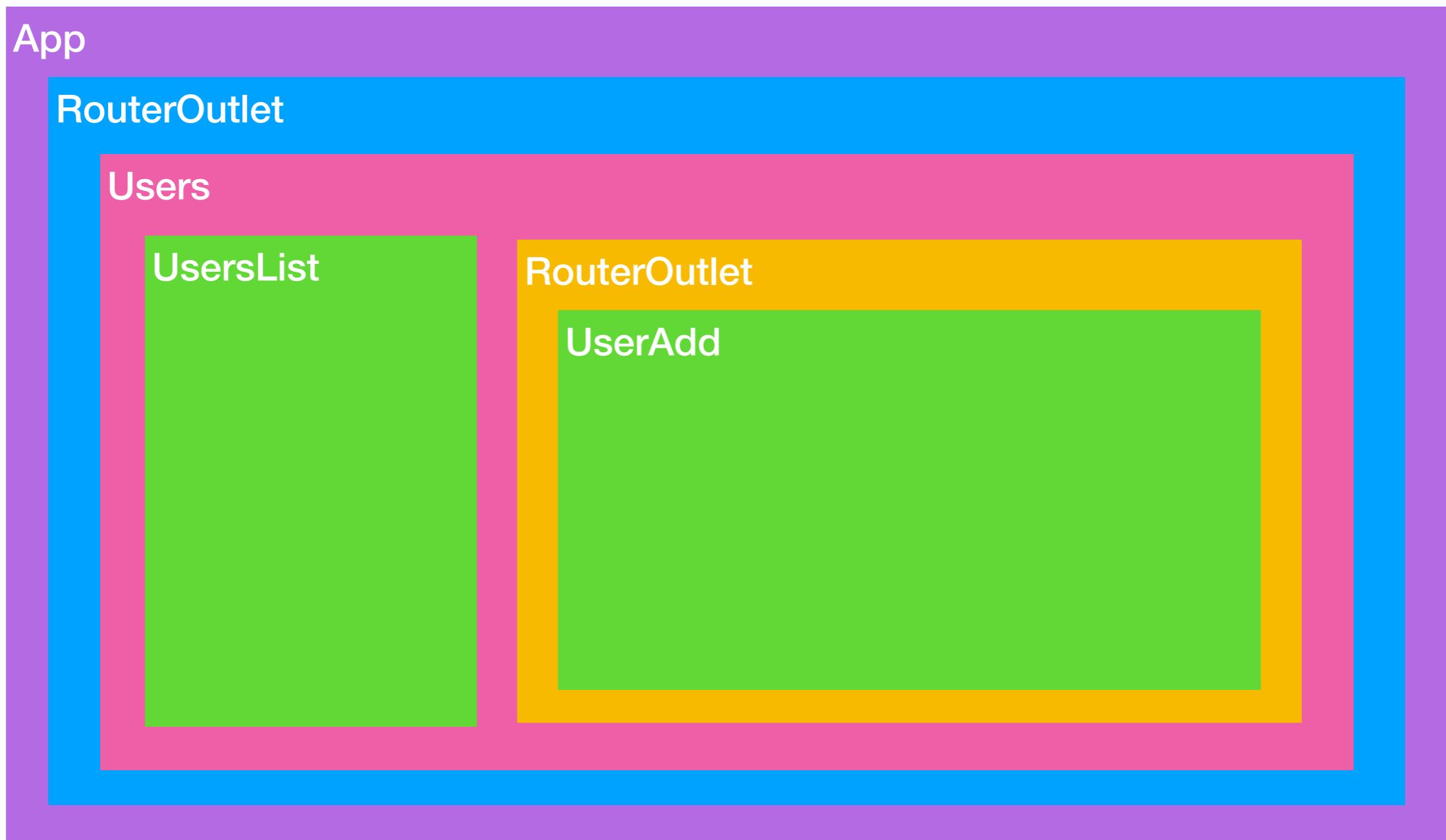
- C'est également via ce service qu'on peut écouter des événements lors des changements de routes
- Liste des événements et leur description :  
<https://angular.io/guide/router#router-events>

```
export class AppComponent implements OnInit {  
  constructor(  
    protected router: Router,  
    protected title: Title,  
  ) {}  
  
  ngOnInit() {  
    this.router.events.pipe(  
      filter((event) => event instanceof ActivationEnd)  
    ).subscribe((event: ActivationEnd) => {  
      const titleText = event.snapshot.data.title || 'AddressBook';  
      this.title.setTitle(titleText);  
    });  
  }  
}
```



# Routeur - Routes imbriquées

- Plutôt que de définir plusieurs *RouterOutlet* nommés, il est souvent plus pratique des les imbriquer



# Routeur - Routes imbriquées



- Pour définir des routes imbriquées on utilise la clé *children* lors de la définition des routes
- Le composant routé parent devra contenir à nouveau un *RouterOutlet*
- Dans notre exemple */users/add* permettra d'accéder au composant *Users* imbriquant *UserAdd* alors que */users/123* imbriquera *UserShow*

```
const routes: Routes = [{
  path: 'users',
  component: UsersComponent,
  children: [
    { path: 'add',
      component: UserAddComponent,
    }, {
      path: ':id',
      component: UserShowComponent,
    }
  ]
};
```



# Routeur - Lazy Loading

- Angular propose un mécanisme au niveau du router appelé Lazy Loading
- Permet d'indiquer à webpack de mettre le contenu du module dans un fichier JS séparé qui ne chargera qu'au moment où on navigue vers les routes commençant par le *path*, réduisant ainsi le temps de chargement initial de l'application
- Il faudra être vigilant et retirer tout référence statiques (import) entre les modules existant et le module à "lazy-loader"

```
// Jusqu'à Angular 7
{
  path: 'users',
  loadChildren: './users/users.module#UsersModule'
},
// Depuis Angular 8
{
  path: 'users',
  loadChildren: () => import('./users/users.module').then(mod =>
mod.UsersModule),
},
```

# Routeur - Guards



- Les guards sont un mécanisme permettant de bloquer certains comportement du routeur (navigation, lazy-loading) selon certaines conditions (authentification, autorisations...)
- On peut générer un Guard avec la commande  
ng generate guard CHEMIN
- La commande demandera alors quels types de Guard à implémenter
  - *CanActivate* pour bloquer l'accès à un composant router
  - *CanActivateChild* pour bloquer l'accès aux routes enfants
  - *CanLoad* pour empêcher le chargement du fichier en cas de Lazy Loading

```
MacBook-Pro:addressbook-angular romain$ ng generate guard test
? Which interfaces would you like to implement? (Press <space> to select, <a> to toggle all, <i> to
invert selection)
>○ CanActivate
○ CanActivateChild
○ CanLoad
```

- Il existe également *CanDeactivate* au moment où l'utilisateur quitte la page

# Routeur - Guards



- Exemple de Guard

```
@Injectable()
export class AuthenticationGuard implements CanActivate, CanActivateChild {
  constructor(private router: Router, private userService: UserService) {}

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot,
  ): Observable<boolean> | Promise<boolean> | boolean {
    return this.userService.currentUsers$.pipe(
      catchError(() => {
        this.router.navigate(['login']);
        return of(false);
      }),
      mapTo(true),
    );
  }
}
```



# Routeur - Resolver

- Un resolver est un Service qui va permettre de "pré-fetcher" les données avant d'afficher le composant routé

```
@Injectable({
  providedIn: 'root',
})
export class UserShowResolverService implements Resolve<User> {
  constructor(private userService: UserService, private router: Router) {}

  resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot):
  Observable<User> | Observable<never> {
    let id = route.paramMap.get('id');

    return this.userService.getById$(id).pipe(
      take(1),
      mergeMap(crisis => {
        if (crisis) {
          return of(crisis);
        } else { // id not found
          this.router.navigate(['/users']);
          return EMPTY;
        }
      })
    );
  }
}
```

# Routeur - Resolver



- La route enregistre alors le *Resolver*

```
{  
  path: ':id',  
  component: UserShowComponent,  
  resolve: {  
    user: UserShowResolverService  
  }  
}
```

- Et les données sont disponibles pour le composant à la clé *data* de *ActivatedRoute*

```
ngOnInit() {  
  this.activatedRoute.data  
    .subscribe((data: { user: User }) => {  
      this.user = data.user;  
    });  
}
```



**formation.tech**

# HttpClient



# HttpClient - Introduction

- Angular a connu 2 classes pour envoyer des requêtes AJAX : *Http* et *HttpClient* (4.3+)
- Depuis Angular 8 il ne reste que *HttpClient* enregistré dans *HttpClientModule* de `@angular/common/http`
- Il faut penser à importer *HttpClientModule* dans *AppModule*

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```



# HttpClient - Service

- Le service HttpClient est ensuite injectable dans l'application
- Il contient une méthode *request*, ainsi que 7 méthodes raccourcies comme *get* et *post*, et une méthode pour les requêtes *jsonp*

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { User } from './user.model';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class UserService {
  constructor(protected httpClient: HttpClient) { }

  getAll$(): Observable<User[]> {
    return this.httpClient.get<User[]>('/users');
  }

  getById$(id): Observable<User> {
    return this.httpClient.get<User>('/users/' + id);
  }

  create$(user: User): Observable<User> {
    return this.httpClient.post<User>('/users', user);
  }
}
```



# HttpClient - Intercepteurs

- Un intercepteur est un service qui va exécuter du code en amont ou en aval d'une requête
- Exemple :
  - Ajout du token d'authentification aux entêtes de la requêtes
  - Gestion des erreurs du backend
  - Base URL de l'API REST

```
@Injectable()
export class ApiBaseUrlInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler):
    Observable<HttpEvent<any>> {
    if (!req.url.startsWith('/assets') && !req.url.startsWith('http')) {
      req = req.clone({
        url: environment.apiBaseUrl + req.url,
      });
    }
    return next.handle(req);
  }
}
```



# HttpClient - Intercepteurs

- Un intercepteur est un service qui va exécuter du code en amont ou en aval d'une requête
- Exemple :
  - Ajout du token d'authentification aux entêtes de la requêtes
  - Gestion des erreurs du backend
  - Base URL de l'API REST

```
@Injectable()
export class ApiBaseUrlInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler):
    Observable<HttpEvent<any>> {
    if (!req.url.startsWith('/assets') && !req.url.startsWith('http')) {
      req = req.clone({
        url: environment.apiUrl + req.url,
      });
    }
    return next.handle(req);
  }
}
```



# HttpClient - Intercepteurs

- On enregistre les intercepteurs dans la section provider de AppModule

```
providers: [
  {
    provide: HTTP_INTERCEPTORS,
    multi: true,
    useClass: ApiBaseUrlInterceptor,
  },
],
```



**formation.tech**

# Formulaires

# Formulaires - Introduction



- Angular contient un module `FormsModule` pour la gestion des formulaires
- Les formulaires peuvent être gérés côté template (Template Driven Form) ou côté TypeScript (Reactive Form)
- Inclure `FormsModule` modifier le comportement des formulaires, il n'est plus nécessaire d'appeler `event.preventDefault()`



# Formulaires - ngModel

- NgModel est une directive qui permet le Data Binding d'une propriété ou sous propriété dans les 2 sens

```
<input type="tel" class="form-control" name="phone" [(ngModel)]="user.phone">
```

- Dans un formulaire la clé name est obligatoire



# Formulaires - Validation

- Pour valider un formulaire ou utiliser des directives comme RequiredValidator ou EmailValidator
- On peut ensuite récupérer l'instant de ngForm ou ngModel en connaissant leur clé `exportAs`

```
<form #form="ngForm" (submit)="form.valid && createUser()">

  <div class="alert alert-danger"
    *ngIf="form.submitted && form.invalid">
    This form contains errors
  </div>

  <div class="form-group">
    <label>Name</label>
    <input type="text" class="form-control" name="name"
      [(ngModel)]="user.name" required #name="ngModel"
      [class.is-invalid]="(form.submitted || name.touched) && name.invalid">
    <div class="invalid-feedback"
      *ngIf="(form.submitted || name.touched) && name.invalid">
      Name is required
    </div>
  </div>

  <button class="btn btn-primary">Add</button>

</form>
```



**formation.tech**

# React

# React - Installation



- Pour mettre en place rapidement un environnement React fonctionnel, on peut utiliser le package `create-react-app`
- Installation avec npm :  
`npm install -g create-react-app`
- Installation avec Yarn :  
`yarn add create-react-app`
- Pour créer le projet :  
`create-react-app NOM_DU_DOSSIER`
- Créer le projet directement  
`npx create-react-app NOM_DU_DOSSIER`  
`npm init react-app NOM_DU_DOSSIER`  
`yarn create react-app NOM_DU_DOSSIER`

# React - Premier composant



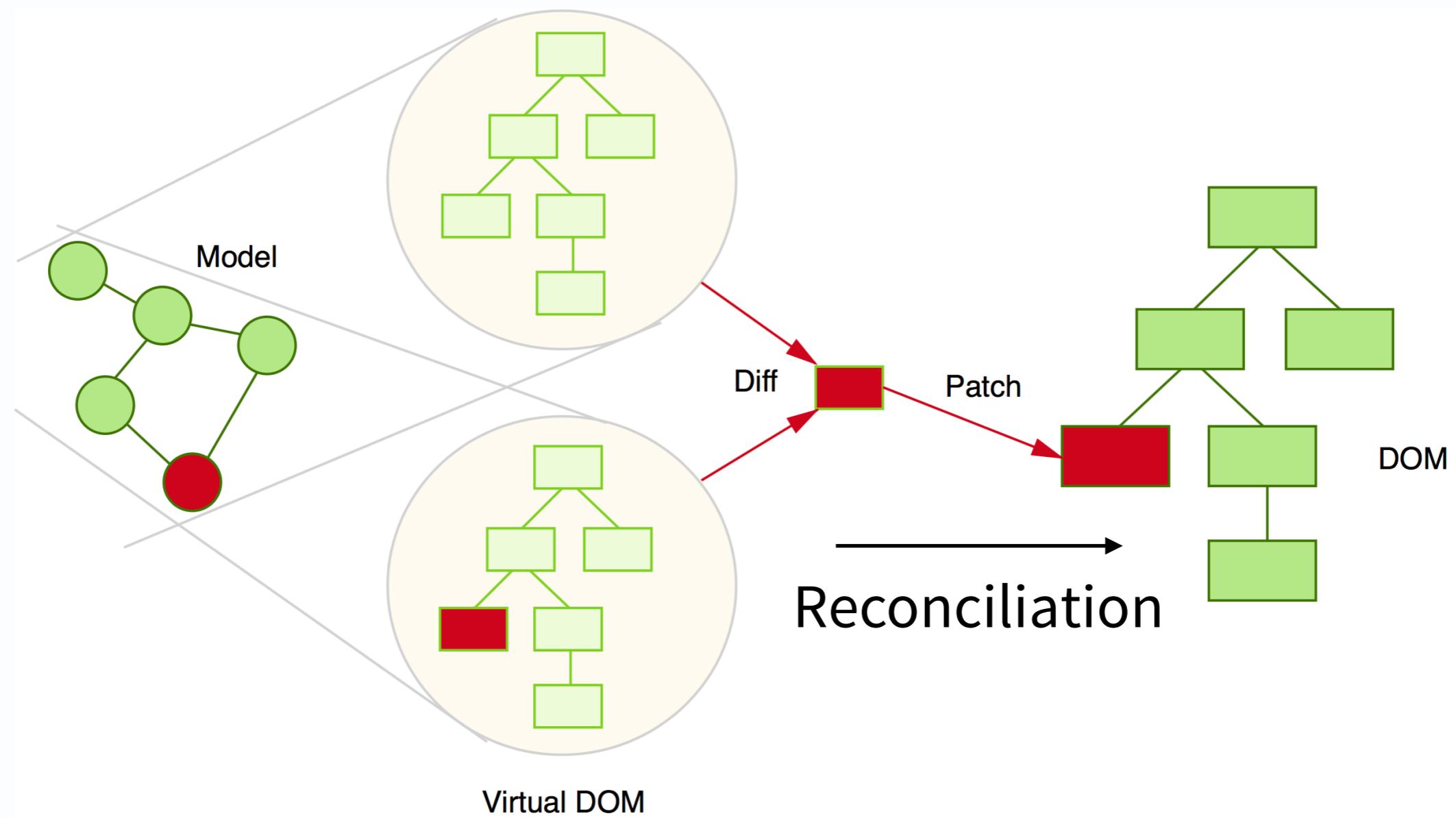
- 2 dépendances :
  - react : permet la création de composants
  - react-dom : permet le rendu de ces composants dans le contexte du DOM
- Notre premier composant App est une simple fonction, qui retourne une syntaxe proche du HTML appelée JSX (l'import de React est obligatoire dans ce cas)
- Par convention les composants React commencent par une majuscule (si vous ne le faites pas, React voit du HTML)

```
import React from 'react';
import ReactDOM from 'react-dom';

const App = () => <h1 className="my-app">Hello</h1>

ReactDOM.render(
  <App />,
  document.getElementById('root'),
);
```

# React - Virtual DOM



# React - JSX



- › Documentation  
<https://facebook.github.io/jsx/>
- › Language proche du HTML nécessitant une compilation (avec Babel par exemple et son plugin babel-plugin-transform-react-jsx)
- › Exemple précédent sans JSX :

```
import React from 'react';
import ReactDOM from 'react-dom';

ReactDOM.render(
  React.createElement('h1', {className: 'my-app'}, 'Hello'),
  document.getElementById('root'),
);
```

# React - JSX



- Conditions en JSX

- if simple

```
import React from 'react';

export const Todo = (props) => {
  return (
    <div>
      <input value={props.value} />
      {props.isDeletable && <button>-</button>}
    </div>
  );
};
```

- if ... else

```
import React from 'react';

export const Todo = (props) => {
  return (
    <div>
      <input value={props.value} />
      {(props.isDeletable) ? <button>-</button> : <button disabled>-</button> }
    </div>
  );
};
```



- Listes en JSX

```
import React from 'react';

export const TodoList = (props) => {
  const listItems = props.todos.map((val, i) =>
    <div key={i}>{val}</div>
  );
  return <div>{listItems}</div>;
};
```

- L'attribut key est obligatoire
  - Il permet à React de savoir si cet élément de la liste doit être ou non rafraîchi.
  - Idéalement une clé id d'un Enregistrement ou Document de base de données.
- Bonne pratique sinon : générer un id avec uuid par exemple.

# React - Stateful components



- Autant les composants les plus simple peuvent être de simple fonction comme vu précédemment, autant la plupart du temps il faudra créer des fonctions constructeurs JS (class en ES6).
- Ces composants auront la possibilité d'entrer en interaction avec d'autres fonctions et leur « state ».
- A minima, écrire une classe qui hérite de `React.Component` et qui implémentent une méthode `render` retournant un sous-arbre JSX.

```
import React from 'react';
import ReactDOM from 'react-dom';

class App extends React.Component {
  render() {
    return <h1 className="my-app">Hello</h1>
  }
}

ReactDOM.render(
  <App />,
  document.getElementById('root'),
);
```

# React - Stateful components



- Sur plusieurs lignes :

```
import React from 'react';
import ReactDOM from 'react-dom';

class App extends React.Component {
  render() {
    return (
      <div>
        <h1 className="my-app">Hello</h1>
        <p>World</p>
      </div>
    );
  }
}

ReactDOM.render(
  <App />,
  document.getElementById('root'),
);
```

- Des parenthèses sont obligatoires si le JSX ne commence pas sur la même ligne que le return.
- Un composant doit avoir un seul élément racine (ici <div>), depuis React 16 on peut retourner un tableau et depuis React 16.2 on peut retourner un Fragment (voir doc)

# React - Propriétés



- Les propriétés ou props, permettent de passer des valeurs au moment du rendu du composant (syntaxe proche d'un attribut HTML)
- Pour accéder à une propriété depuis le composant on passe par sa propriété props (ici en JSX {this.props.content} )

```
import React from 'react';
import ReactDOM from 'react-dom';

class App extends React.Component {
  render() {
    return <h1 className="my-app">{this.props.content}</h1>
  }
}

ReactDOM.render(
  <App content="Hello props"/>,
  document.getElementById('root'),
);
```

# React - Propriétés



- Définir la typologie et la validation des propriétés du composants
- Installer prop-types (voir aussi airbnb-prop-types)  
npm install prop-types

```
import PropTypes from 'prop-types';

class Contact extends React.Component {
  render() {
    return <p>
      Hello my name is {this.props.name},
      I'm {this.props.age}
    </p>;
  }
}

Contact.propTypes = {
  name: PropTypes.string.isRequired,
  age: PropTypes.number,
};
```



# React - Propriétés

- Validateurs personnalisés :

```
Contact.propTypes = {
  name: PropTypes.string.isRequired,
  age(props, propName, componentName) {
    if (props[propName] && (props[propName] < 0 || props[propName] > 120)) {
      return new Error(`$ {propName} should be higher than 0 and lower than 120`)
    }
  },
};
```

- Autres validateurs possibles :

<https://github.com/facebook/prop-types>

- Valeurs par défaut :

```
Contact.defaultProps = {
  name: 'John'
};
```

# React - Refs



- Référencer des éléments du DOM avec refs

```
class ContactAdd extends React.Component {
  add(e) {
    e.preventDefault();
    console.log(this.refs.prenom.value);
    console.log(this.refs.nom.value);
  }
  render() {
    return <form onSubmit={this.add.bind(this)}>
      <div>
        Prénom : <input ref="prenom" />
      </div>
      <div>
        Nom : <input ref="nom" />
      </div>
      <button>+</button>
    </form>;
  }
}
```

# React - State



- Props permet de communiquer avec le composant, state est son état interne, la méthode render est appelée à chaque modification
- On ne peut pas modifier le state directement, il faut utiliser la méthode setState

```
class CounterButton extends React.Component {  
  constructor() {  
    super();  
    this.state = {  
      count: 0,  
    };  
  }  
  increment() {  
    this.setState({  
      count: this.state.count + 1,  
    });  
  }  
  render() {  
    return <button onClick={this.increment.bind(this)}>  
      {this.state.count}  
    </button>;  
  }  
}
```

# React - State



- Il n'est pas nécessaire de modifier tout le state à chaque appel de setState
- Pour des questions de performance, les objets et tableaux du state seront de préférence immuables

```
export class Todo extends Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      saisie: '',  
      liste: []  
    };  
  }  
  
  inputHandler(e) {  
    this.setState({  
      saisie: e.target.value,  
    })  
  }  
  
  formHandler(e) {  
    this.setState((prevState) => ({  
      liste: [...prevState.liste, this.state.saisie]  
    }))  
  }  
  // ...  
}
```

# React - Imbrication de composants



- Lorsque qu'un state doit être accessible par plusieurs composant, il faut le définir sur le plus proche ancêtre commun, les composants imbriqués y accéder au travers de props

```
class App extends React.Component {
  constructor() {
    super();
    this.state = { count: 0 };
  }
  increment() {
    this.setState({ count: this.state.count + 1 });
  }
  decrement() {
    this.setState({ count: this.state.count - 1 });
  }
  render() {
    return <div className="App">
      <h1>{this.state.count}</h1>
      <CounterButton update={this.increment.bind(this)}>+</CounterButton>
      <CounterButton update={this.decrement.bind(this)}>-</CounterButton>
    </div>;
  }
}

class CounterButton extends React.Component {
  render() {
    return <button onClick={this.props.update}>
      {this.props.children}
    </button>;
  }
}
```

# React - Formulaires



```
import React, { Component } from 'react';

export class ContactAdd extends Component {
  constructor(props) {
    super(props);
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(e) {
    this.setState({
      [e.target.name]: e.target.value
    });
  }

  handleSubmit(e) {
    e.preventDefault();
    // fetch()...
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <div className="form-group">
          <label>Prénom</label>
          <input type="text" className="form-control" name="firstName" onChange={this.handleChange} />
        </div>
        <div className="form-group">
          <label>Name</label>
          <input type="text" className="form-control" name="lastName" onChange={this.handleChange} />
        </div>
        <button type="submit" className="btn btn-default">Add</button>
      </form>
    );
  }
}
```

# React - Cycle de vie



- Chaque composant à un certain nombre de méthodes liées à son cycle de vies :
- Chargement
  - `constructor()`
  - ~~`componentWillMount()`~~ (dépréciée)
  - `render()`
  - `componentDidMount()`
- Destruction
  - `componentWillUnmount()`
- Mise à jour
  - ~~`componentWillReceiveProps()`~~ (dépréciée)
  - `shouldComponentUpdate()`
  - ~~`componentWillUpdate()`~~ (dépréciée)
  - `render()`
  - `componentDidUpdate()`
- <https://reactjs.org/blog/2018/03/27/update-on-async-rendering.html>

# React - Cycle de vie



- `constructor()`
  - Appelée côté client et serveur
  - `constructor` sert à initialiser state et props

# React - Cycle de vie



- `componentDidMount()`
  - Le rendu initial du composant a été effectué
  - Il est possible de manipuler le DOM
  - N'existe que côté client
  - Le bon endroit pour charger un plugin jQuery ou tout ce qui ne s'exécute qu'une seul fois et qui a besoin d'accéder au DOM
  - Démarrer des requêtes AJAX, des timers

# React - Cycle de vie



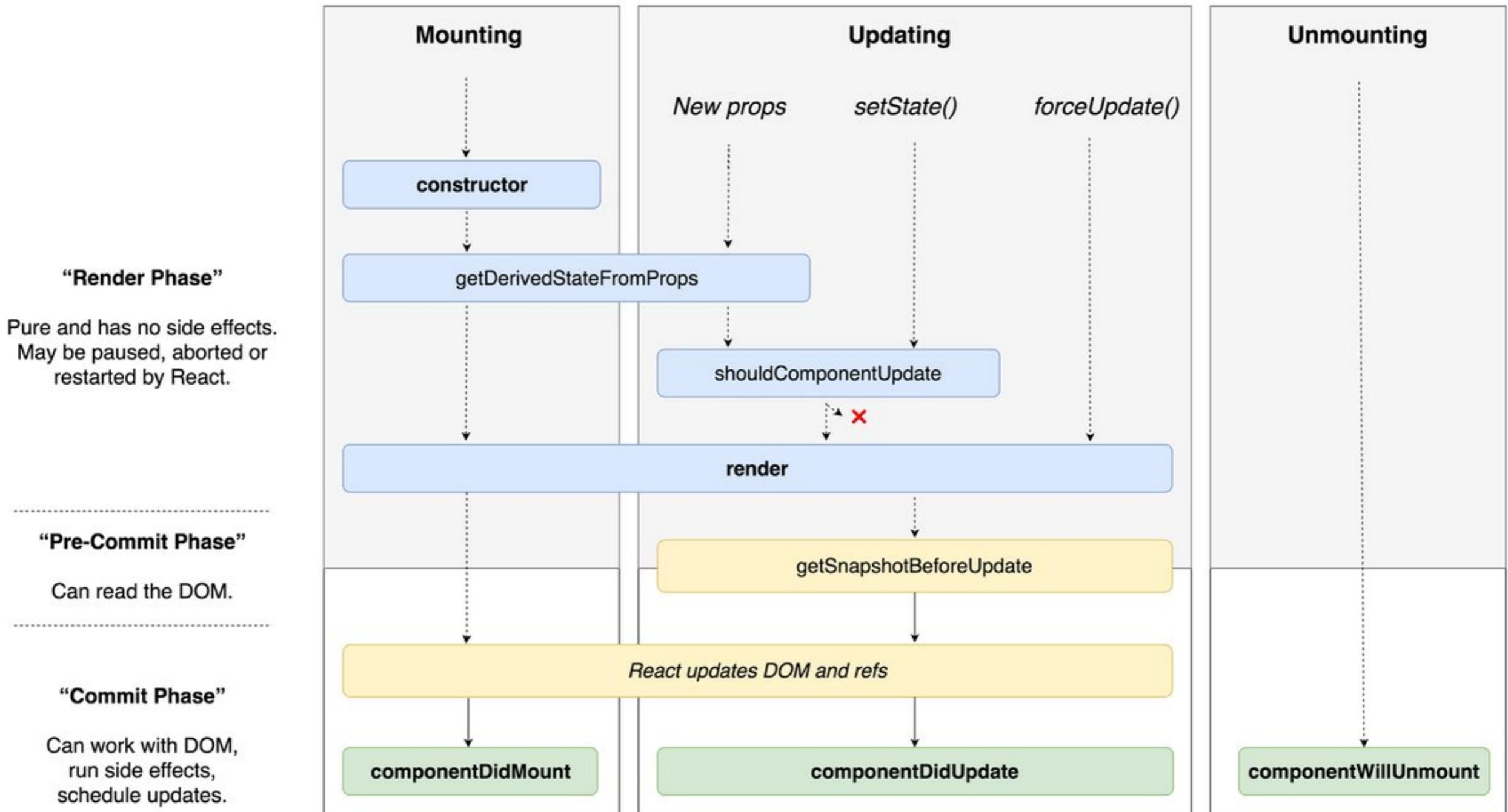
- `shouldComponentUpdate()`
  - Permet d'empêcher un `render()`, doit répondre true ou false
  - Utile pour optimiser une application, ne pas faire de rendu si les props ou le state on été modifié d'une manière qui ne nécessite pas un nouveau rendu (voir aussi `PureComponent`)

# React - Cycle de vie



- `componentDidUpdate()`
  - Juste après un rendu autre que initial
  - On a accès au DOM
  - Le bon endroit pour un update d'un plugin jQuery (Chosen, Select2...)
- `componentWillUnmount()`
  - Le composant va être supprimer
  - Permet de supprimer des listeners, libérer la mémoire, appeler clearInterval/Timeout, sinon l'objet associé au composant ne sera jamais détruit (si ref interne dans un callback)

# React - Cycle de vie



# React - Higher Order Components



- Un Higher Order Component (HOC) est une fonction qui reçoit un composant en entrée et retourne un nouveau composant (une liste filtrée, remplie, etc...)
- Exemple, connect dans react-redux, qui injecte la fonction dispatch à un composant
- Voir Recompose (bibliothèque d'utilitaire HOC)  
<https://github.com/acdlite/recompose/>

```
export default connect()(TodoApp);
```

# React - Higher Order Components



- Ajouter de nouvelles props via un HOC

```
render() {
  // Filter out extra props that are specific to this HOC and shouldn't be
  // passed through
  const { extraProp, ...passThroughProps } = this.props;

  // Inject props into the wrapped component. These are usually state values or
  // instance methods.
  const injectedProp = someStateOrInstanceMethod;

  // Pass props to wrapped component
  return (
    <WrappedComponent
      injectedProp={injectedProp}
      {...passThroughProps}
    />
  );
}
```

# React - Higher Order Components



- Renommer le composant retourné (bonne pratique)

```
import React, { Component } from 'react';

function getDisplayName(WrappedComponent) {
  return WrappedComponent.displayName || WrappedComponent.name || 'Component';
}

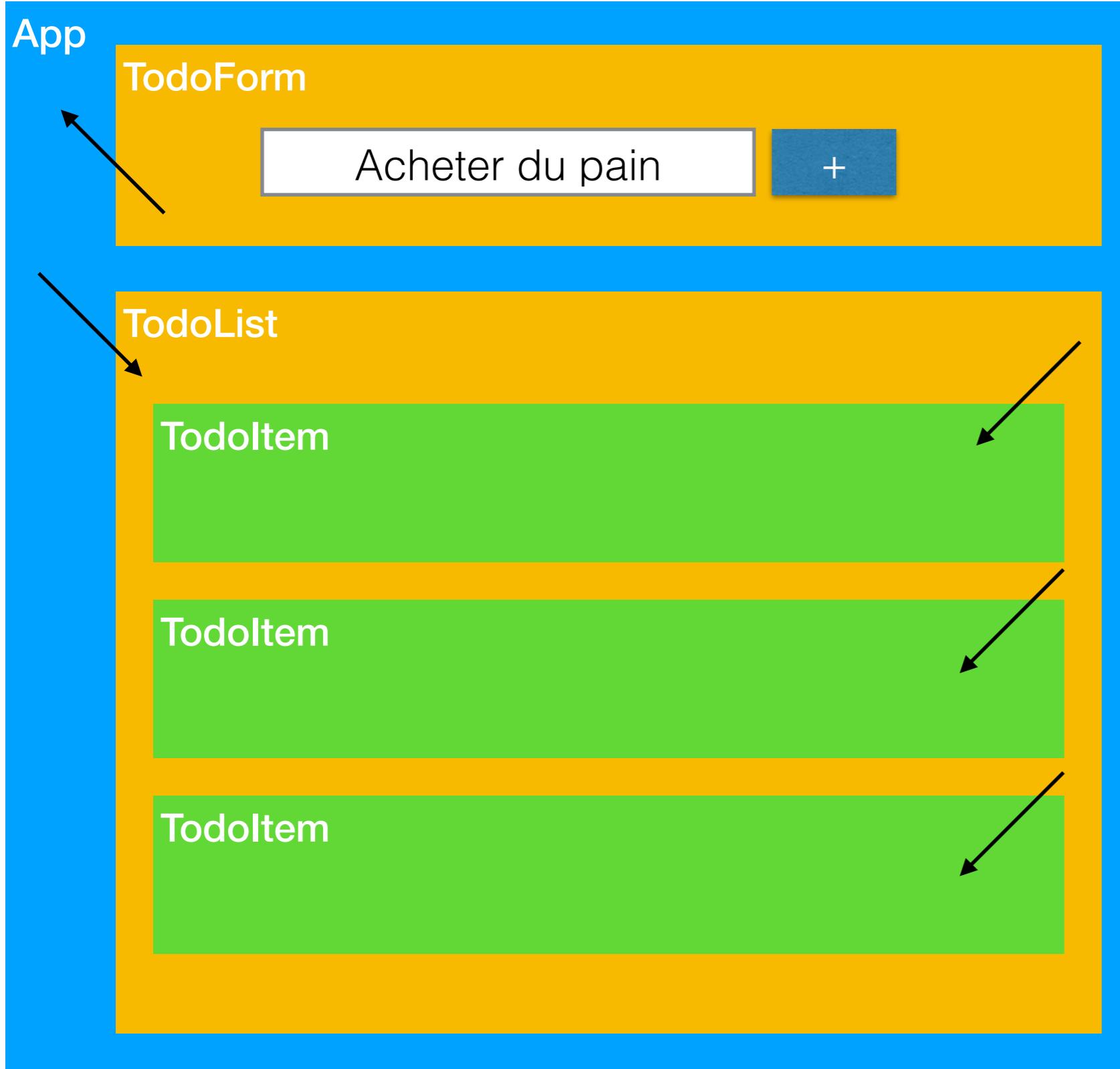
export const logLifecycle = (WrappedComponent) => {

  class LogLifecycle extends Component {
    // ...
  }

  LogLifecycle.displayName = `LogLifecycle(${getDisplayName(WrappedComponent)})`;

  return LogLifecycle;
};
```

# React - Exercice



- Créer un projet todo-react avec `create-react-app`
- Créer 3 composants :
  - **TodoForm**
  - **TodoList**
  - **TodoItem**
- Dans le state de **App** créer une liste de todos (`string[]` ou `object[]`)
- Passer ce tableau à **TodoList**, qui créera autant de **TodoItem** qu'il y a d'élément dans le tableau
- **TodoItem** reçoit un élément et l'affiche
- **TodoForm** Controlled Component, reçoit un callback de **App** et lui passe la valeur saisie au submit du form
- Le callback de **App** ajoute l'élément au tableau