

JS

1. Rappels
2. Environnement de debug
- 3. ES6+**
4. POO
5. jQuery
6. Expressions régulières
7. Échange de données
8. APIs HTML5

3. ES6+

- **ECMAScript ?**
 - Les principales nouveautés
 - Babel
 - Les modules

ECMAScript (ES)

- Spec gérée par ECMA (ex. European Computer Manufacturers Association)
- Google, Intel, IBM, Microsoft, etc.
- Spécification de langage de script : ECMA-262
- Suivie par JavaScript mais aussi ActionScript (Flash/Flex) et JScript (IE3-9)
- groupe de travail : TC39 (Technical Committee 39)

Notes :

La liste des membres de ECMA peut se trouver sur le site de l'organisation : <http://www.ecma-international.org/memento/members.htm>

ECMAScript n'est qu'une des spécifications gérées par ECMA. On note les formats JSON (ECMA-404), Open XML (ECMA-388), le langage C# (ECMA-334), ou encore la spécification des volumes FAT12/16 (ECMA-107) et des CD-ROM ! (ECMA-119) cf. https://en.wikipedia.org/wiki/List_of_Ecma_standards

Les compte-rendus des réunions du TC39 sont accessibles en ligne sur github : <http://tc39.github.io/tc39-notes/>

cf. <http://2ality.com/2015/11/tc39-process.html>

ES : UN RYTHME IRRÉGULIER 1.4

- 1995 : Netscape LiveScript
- 1996 : renommé en JavaScript et soumis à ECMA
- 1997 : ES1
- 1998 : ES2
- 1999 : ES3
- ...
- 2008 : abandon de ES4
- 2009 : sortie de ES5 (ES3.1)
- ...
- juin 2015 : ES6 / ES2015
- juin 2016 : ES7 / ES2016
- juin 20... : ES... / ES20...
- juin 2020 : ES11 / ES2020

Notes :

Plus d'infos sur : <http://www.benmvp.com/learning-es6-history-of-ecmascript/>

Nouveau rythme depuis ES6/ES2015 :

- TC39 process : une version tous les ans depuis ES6
- nouvelle numérotation : ES20XX
- objectif : éviter les mises à jours trop lourdes (comme ES6)

ES : UN NOUVEAU CYCLE

1.5

plusieurs stades de normalisation :

stage 0 : strawman	Idée non formalisée
stage 1 : proposal	Proposition formelle et argumentée
stage 2 : draft	Première version de la spécification
stage 3 : candidate	En attente de retour d'implémentation
stage 4 : finished	Sera intégrée à la prochaine norme

Notes :

- stage 0 : doit provenir d'un membre du TC39 ou un contributeur enregistré
- stage 1 : proposition doit expliquer le problème qu'elle résout, proposer des exemples de code et une proposition d'API. Identifie les obstacles éventuels, les polyfills possibles etc.
- stage 2 : à ce stade la spec a de grandes chances de se retrouver dans le standard
- stage 3 : la feature doit avoir été implémentée et testée au minimum 2 fois
- stage 4 : la feature est validée et sera intégrée à la prochaine yearly release

cf. <https://tc39.github.io/process-document/> La liste des features d'ECMAScript en cours de spécification et leur stage peut se consulter sur le repo github du TC39 : <https://github.com/tc39/proposals>

3. ES6+

- ECMAScript ?
- **Les principales nouveautés**
- Babel
- Les modules

ES6 : LET

- remplace "var"
- scopée

```
let i = 0;  
for ( let i = 1; i < tableau.length; i++ ) {  
    console.log( i ); // 1, 2, 3, ...  
}  
console.log( i ); // 0
```

Notes :

Fonctionne exactement comme le `var` traditionnellement employé. Cependant, le `let` a la particularité d'être scopé au niveau du bloc de code dans lequel il est déclaré, et pas forcément dans toute la fonction.

ES6 : CONST

- déclaration de constante
- portée

```
const hello; // Erreur : une constante doit avoir une valeur !
const hello = 'Hello';

if (true) {
  const hello = 'Bonjour';
  hello = 'Hola'; // Erreur : une constante ne peut être modifiée !
  console.log(hello); // "Bonjour"
}

console.log(hello); // "Hello"
```

Notes :

`const` permet de déclarer une constante. De plus, comme le `let`, une variable `const` est scopée au niveau du bloc de code et non pas de la fonction comme le `var`.

ES6 : CONST ?

```
const ingredients; // 🚫 Erreur : une constante doit avoir une valeur
```

```
const ingredients = ['methanol', 'red phosphorus']; // 👍
```

```
ingredients = ['methanol', 'red phosphorus', 'pseudo']; // 🚫 Erreur :  
// réassignation impossible
```

```
ingredients.push('pseudo'); // 👍
```

ES6 : TEMPLATES STRINGS

- Nouveau délimiteur de chaîne : `
- Injection d'expressions (variable ou appel de fonction)
- multiline

```
const value = `danger`;
let phrase = `I am the ${ value } !`; // I am the danger !

function myFunction(valeur) {
    return valeur;
}
phrase = `I am the ${ myFunction('danger') } !`; // I am the danger !
```

```
function renderLink( target, cssClass, href ) {
    return `<a href="${ href.toLowerCase() }"
        class="${ cssClass }"
        target="${ target }">
        Cliquez ici !
    </a>`; // multiline !
}
```

ES6 : DESTRUCTURING

- déclarer des variables au nom d'une propriété d'objet
- ... ou qui correspondent aux valeurs d'un tableau

```
const personnage = { prenom: 'Skyler', nom: 'White' };  
// const prenom = personnage.prenom, nom = personnage.nom;  
const { prenom, nom } = personnage;  
console.log(`Salut ${prenom} ${nom} !`);
```

```
function maFonction( {prenom, nom} ) {  
    return `Salut ${prenom} ${nom} !`;  
}  
maFonction( personnage );
```

```
const tableau = [ 'Walter Jr', 'White' ];  
// const prenom = tableau[0], nom = tableau[1];  
const [ prenom, nom ] = tableau;
```

Notes :

Le destructuring permet de faciliter la création et l'assignation de variables à partir issues d'objets ou de tableaux.

<http://codepen.io/kumquats/pen/dXWxzW?editors=0011>

ES6 : ARROW FUNCTIONS

1.12

- déclaration de fonction simplifiée
- return implicite
- scope préservé

```
// Fonction anonyme en ES5
var add = function( a, b ) {
    return a + b;
}

// et en ES6 (opérateur "fat arrow")
const add = ( a, b ) => a + b;
const square = x => x * x;
// si un seul paramètre, pas besoin de parenthèses
```

Notes :

La valeur de `this` dans une arrow function est toujours celui dans lequel elle est déclarée :

```
const character1 = {
  name: 'hank',
  link: document.querySelector( 'a.link' ),
  init: function(){
    this.link.addEventListener( 'click',
      (event)=>{
        event.preventDefault();
        console.log(this.name);
      }
    );
  }
}
character1.init();
```

cf. <https://codepen.io/uidlt/pen/wvGLwmL?editors=1001>

ES6 : VALEURS PAR DÉFAUT

Valeurs par défaut pour les paramètres des fonctions

```
function killRandomHero( lastname = 'Stark' ){  
  const c = getCharacter(lastname);  
  c.isDead = true;  
}
```

```
// Equivalent en ES5  
function killRandomHero( lastname ){  
  if ( lastname === undefined ) {  
    lastname = 'Stark';  
  }  
  // ou  
  lastname = lastname || 'Stark';  
  const c = getCharacter(lastname);  
  c.isDead = true;  
}
```

3. ES6+

- ECMAScript ?
- Les principales nouveautés
- **Babel**
- Les modules



PROBLÈME : SUPPORT NAVIGATEUR

1.15

ECMAScript 2015 (ES6)

Support for the ECMAScript 2015 specification. Features include Promises, Modules, Classes, Template Literals, Arrow Functions, Let and Const, Default Parameters, Generators, Destructuring Assignment, Rest & Spread, Map/Set & WeakMap/WeakSet and many more.

IE	Edge	Firefox	Chrome	Safari	iOS Safari	Opera Mini	Chrome for Android	Android Browser	Samsung Internet
9	87	85	87	13	13.3			4.4	11.2
10	88	86	88	13.1	13.7			4.4.4	12.0
11	89	87	89	14	14.5	all	89	89	13.0
		88	90	TP					



 Partial Support

Global: 94.41% + 2.94% = 97.35%

Data from caniuse.com | Embed from caniuse.bitsofco.de


6 April 2021

Notes :


Le support navigateur est très partiel, notamment sur IE. Il faut noter que chaque feature ES6 dispose de son propre support navigateur de manière indépendante. <http://caniuse.com/#search=es6>

<http://kangax.github.io/compat-table/es6/> Autre tableau détaillé, navigateur par navigateur du support des différentes fonctionnalités ES6.

Pour ES2016 et suivantes voir <http://kangax.github.io/compat-table/es2016plus/> et pour les features encore en cours de normalisation : <http://kangax.github.io/compat-table/esnext/>


 **7.10.0**

DocsSetupTry it outVideosBlog

 Search

DonateTeam

1,177 GitHub

GET BABEL HOLIDAY APPAREL 


Babel is a JavaScript compiler.

Use next generation JavaScript, today.

Babel 7 is out! Please read our [announcement](#) and [upgrade guide](#) for more information.

Put in next-gen JavaScript	Get browser-compatible JavaScript out
<pre>let yourTurn = "Type some code in here!";</pre>	<pre>let yourTurn = "Type some code in here!";</pre>

Special Sponsor



handshake

Decentralized certificate authority and naming

Notes :

BabelJS <https://babeljs.io/>

Babel est un compilateur de code ES6+ en ES5.

Cet outil permet de rendre notre application JS compatible avec les vieux navigateurs.

Le principe est le suivant :

- on code de manière élégante à l'aide des dernières features du langage (ES6+)
- Babel compile le tout en ES5 (compatible avec les anciens navigateurs),
- c'est ce fichier compilé qui sera mis en ligne et servi aux visiteurs

BABEL

DocsSetupTry it outVideosBlogSearchDonateTeam118GitHub

```
1 // Arrow function
2 let square = x => x * x;
3
4 // const
5 const i = square(42);
6 //i = 12; //erreur de compilation (constante)
7
8 // let scopé
9 for ( let i = 0; i < 12 ; i++ ){
10   console.log(i);
11   if (true){
12     let i = 5;
13     console.log(i);
14   }
15 }
16
17 // destructuring
18 o = {
19   a: "a",
20   b: "b",
21 };
22 let {a, b} = o;
23
24 // objet rest spread (stage 3)
25 const params = { lol: 'lol' };
26 const fusion = {
27   pouet: 'pouet',
28   ...params
29 }
30
```

```
1 "use strict";
2
3 // Arrow function
4 var square = function square(x) {
5   return x * x;
6 }; // const
7
8
9 var i = square(42); //i = 12; //erreur de compilation (constante)
10 // let scopé
11
12 for (var _i = 0; _i < 12; _i++) {
13   console.log(_i);
14
15   if (true) {
16     var _i2 = 5;
17     console.log(_i2);
18   }
19 } // destructuring
20
21
22 o = {
23   a: "a",
24   b: "b"
25 };
26 var _o = o,
27   a = _o.a,
28   b = _o.b; // objet rest spread (stage 3)
29
30 var params = {
31   lol: 'lol'
32 };
33 var fusion = {
34   pouet: 'pouet',
35   ...params
36 };
```

Notes :

Babel dispose d'un outil en ligne qui permet de tester la conversion de code ES6+ en ES5. Cet outil a surtout un but pédagogique : montrer comment les syntaxes ES6+ permettent d'avoir un code plus propre et plus lisible que son équivalent en ES5.

[Outil de test en ligne du compilateur babel \(repl\)](#)

[babel repl pré-rempli avec des exemples de code ES6](#)

3. ES6+

- ECMAScript ?
- Les principales nouveautés
- Babel
- **Les modules**

MODULES : PROBLÉMATIQUE

1.21

```
<!-- Sans utilisation des modules -->  
<script src="header.js"></script>  
<script src="menu.js"></script>  
<script src="breadcrumb.js"></script>  
<script src="footer.js"></script>  
<script src="main.js"></script>
```



```
<!-- Avec utilisation des modules -->  
<script type="module" src="main.js"></script>
```

Notes :

Le système de modules permet de gérer plus facilement les dépendances entre les différents fichiers de notre application.

Au lieu de lister à plat l'ensemble des fichiers requis par notre application (dans l'ordre adéquat !) c'est le fichier JS principal qui indique les fichiers dont il dépend, qui eux même indiquent les fichiers dont ils dépendent et ainsi de suite.

Ce mécanisme est particulièrement utile lorsque l'on développe des applications utilisant de nombreux fichiers et notamment des librairies tierces.

MODULES : PRINCIPE

vehicle.js

```
const vehicle = 'the RV';  
export default vehicle;
```

main.js

```
import vehicle from './vehicle.js';  
console.log( vehicle ); // 'The RV'
```

Notes :

Le principe des modules est de diviser le code JavaScript en plusieurs fichiers appelés "modules".

Toutes les variables contenues dans un module ne sont par défaut disponibles qu'au sein de ce module. En revanche il est possible d'exporter certaines variables afin de les exposer au reste de l'application.

Un module peut ainsi récupérer les valeurs exportées depuis un autre module grâce à l'instruction "import".

MODULES : EXPORTS MULTIPLES 1.23

vehicle.js

```
const vehicle = 'the RV';
const owner = 'Jesse Pinkman';
export default vehicle;
export const message = 'owner is ' + owner;
```

main.js

```
import vehicle from './vehicle.js';
import { message, owner } from './vehicle.js';
console.log( message, owner ); // 'Owner is Jesse Pinkman', undefined
```

Notes :

Un module peut exporter plusieurs valeurs, mais une seule peut être l'export par défaut.

Pour exporter des valeurs supplémentaires, il suffit d'utiliser l'instruction `export` suivi de la valeur, sans le mot clé `default`.

Au niveau des imports, il y a là aussi une différence : il faut entourer le nom de la (ou des) valeur qu'on veut importer avec des accolades :

```
import { maValeur } from './monModule.js';
```

On peut tout à fait importer plusieurs valeurs avec une seule instruction `import`, il faut alors séparer les valeurs par des virgules :

```
import { maValeur1, maValeur2 } from './monModule.js';
```

Si l'on veut récupérer à la fois l'export par défaut et d'autres exports avec une seule instruction alors on peut écrire :

```
import maValeurParDefaut, { maValeur1, maValeur2 } from
'./monModule.js';
```

JavaScript modules via script tag

Loading JavaScript module scripts (aka ES6 modules) using ``<script type="module">`` Includes support for the ``nomodule`` attribute.

IE	Edge	Firefox	Chrome	Safari	iOS Safari	Opera Mini	Chrome for Android	Android Browser	Samsung Internet
9	87	85	87	13	13.3			4.4	11.2
10	88	86	88	13.1	13.7			4.4.4	12.0
11	89	87	89	14	14.5	all	89	89	13.0
		88	90	TP					

✓ ✗ Partial Support

Global: 92.38% + 0.19% = 92.57%

Data from caniuse.com | Embed from caniuse.bitsofco.de

6 April 2021

Notes :

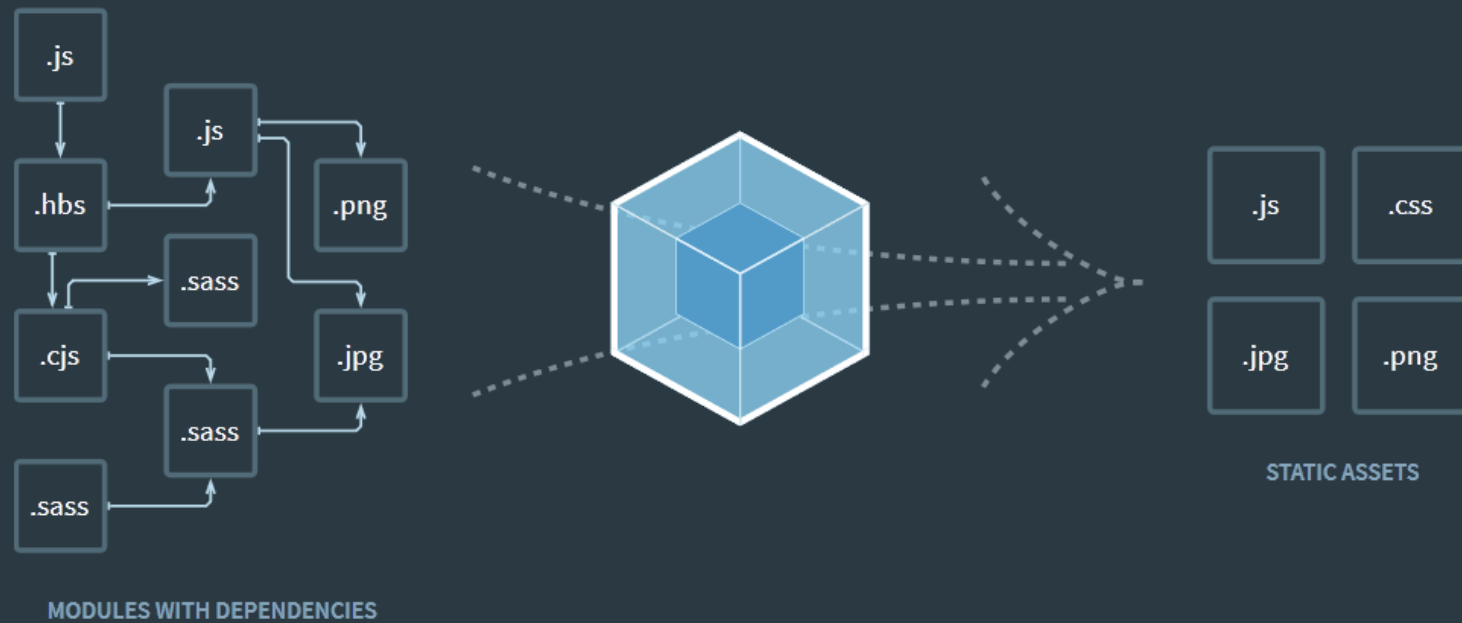
Support navigateur des modules ES6 : <https://caniuse.com/#feat=es6-module>

Le support des modules ES6 est grandissant et tous les navigateurs modernes supportent désormais cette syntaxe.

Mais comme bien souvent, Internet Explorer et Android browser (4.4) viennent gâcher la fête.

Heureusement est arrivé...

bundle your scripts



Write your code

src/index.js

```
import bar from './bar';

bar();
```

src/bar.js

```
export default function bar() {
  //
}
```

Notes :

<https://webpack.js.org/>

webpack permet de regrouper dans un seul fichier (le "bundle") l'ensemble des fichiers nécessaires au fonctionnement de l'application et donc d'étendre le support navigateur au delà des navigateurs qui supportent le système de modules.

Webpack permet d'intégrer différents type de "loaders".

Les loaders sont des filtres qui sont appliqués aux fichiers avant qu'ils ne soient ajoutés dans le bundle.

Le loader le plus fréquemment utilisé est `babel-loader` qui permet de s'assurer que les modules contenant de l'ES6 seront bien compilés en ES5.