

# EcmaScript 6

Ou ECMAScript 2015

# compatibilité

- ECMAScript6 ou 2015 présente quelques unes des dernières évolutions du Javascript.
- Pas compatible avec tous les navigateurs (cf [table de compatibilité](#))

[illegible]

# Outil de compatibilité

- Afin d'assurer la compatibilité entre un programme en ECMAScript 6 et la majorité des navigateurs, on utilise des outils de transposition en ECMAScript 5 tels que [Babel](#) ou Traceur.
- Le code ES6 est ainsi compilé en ES5
- On parle aussi de transpileur



# Babel is a JavaScript compiler.

Use next generation JavaScript, today.

Put in next-gen JavaScript

```
var name = "Guy Fieri";  
var place = "Flavortown";  
  
`Hello ${name}, ready for ${place}?`;
```

Get browser-compatible JavaScript out

```
var name = "Guy Fieri";  
var place = "Flavortown";  
  
"Hello " + name + ", ready for " + place + "?"
```

[Check out the REPL to experiment more!](#)



## Who's using Babel?



facebook



NETFLIX

# installation

- Chaque traducteur ou « transformer » de Babel est un plugin qui peut être installé séparément.
- Babel permet de réaliser différentes « traductions »
- Un « preset » est un groupe de « transformers » qui permet de charger les principaux plugin en une seule passe.

# Const, Let & Var

La fin du Var

# const, let, var

**Const** : permet de déclarer une variable à assignation unique

```
function fn() {  
    const foo = "bar"  
    if (true) {  
        const foo // SyntaxError, la variable doit être assignée  
        const foo = "qux"  
        foo = "norf" // SyntaxError, la variable ne peut pas être réassignée  
        console.log(foo)  
        // "qux", la variable appartient au bloc "if"  
    }  
    console.log(foo)  
    // "bar", la variable appartient à la fonction "fn"  
}
```

# const, let, var

**Const** : Elle n'est pas réassignable, mais elle n'est pas non plus immuable, elle est réévaluable

```
function fn() {  
    const arr = [1, 2, 3];  
    for (const el of arr) {  
        console.log(el);  
    }  
}
```



# const, **let**, var

**Let** : est réassignable et scopée au bloc courant

```
function fn() {  
    let foo = "bar";  
    var foo2 = "bar";  
    if (true) {  
        let foo; // pas d'erreur, foo === undefined  
        var foo2;  
        // Attention, les déclarations "var" ne sont pas scopées au niveau bloc  
        // foo2 est en réalité écrasé !  
        foo = "qux";  
        foo2 = "qux";  
        console.log(foo); // "qux", la variable appartient au scope de son blocs (le "if")  
        console.log(foo2); // "qux"  
    }  
    console.log(foo); // "bar", la variable appartient au scope de son bloc (la fonction "fn")  
    console.log(foo2); // "qux"  
}
```

# const, let, **var**

## **Var** : est réassignable et scopée à la fonction

À noter que JS remonte les déclarations de var au début du scope de la fonction  
On parle de HOISTING.

```
function fn() {  
    console.log(foo); // undefined (au lieu de ReferenceError)  
    var foo = "bar";  
}
```

Ce n'est pas le cas pour **let** et **const**, on parle alors de Temporal Dead Zone

```
function fn() {  
    console.log(foo); // ReferenceError, foo est dans la TDZ  
    let foo = "bar";  
}
```

# Destructuring

assigner des variables  
provenant d'un objet  
en se reposant sur la  
structure

```
var myObject = {  
  foo: 1,  
  bar: 2  
};
```

```
// Avec ES5, vous deviez par exemple faire  
var foo = myObject.foo;  
var bar = myObject.bar;  
foo; // 1  
bar; // 2 //
```

```
Avec ES6, vous pouvez l'écrire sous la forme  
const { foo, bar } = myObject;  
foo; // 1  
bar; // 2
```

# Arrow Functions

Ou fonctions fléchées

Les fonctions fléchées  
offrent une syntaxe  
raccourcie des fonctions  
en utilisant la syntaxe =>

Les fonctions fléchées  
n'ont pas de this, elles  
utilisent celui du parent

```
const myFn = (x) => {  
    return x + 1; };
```

```
// ===
```

```
const myFn = (x) => x + 1;
```

```
// ===
```

```
const myFn = x => x + 1;
```

```
// ===
```

```
const myFn = x => (x + 1);
```

# Classes

Un sucre syntaxique pour les traditionalistes de l'objet

# Avant

- créer le constructeur de cet objet sous forme d'une fonction.
- ajouter toutes les propriétés sur cette fonction.
- remplacer la propriété prototype de ce constructeur par un objet ad hoc.
- Cet objet prototype contiendra toutes les méthodes nécessaires aux instances de notre classe.
- Même pour un exemple très simple, une fois que vous en aurez fini, vous allez vous retrouver avec un code lourd.

```
function Cercle(rayon) {  
    this.rayon = rayon;  
    Cercle.nbrDeCercles++; }  
  
Cercle.dessiner = function dessiner(cercles, canvas)  
{ /* Code pour dessiner dans le Canvas */ }  
Object.defineProperty(Cercle, "nbrDeCercles", {  
    get: function()  
        { return !this._count ? 0 : this._count; },  
    set: function(val)  
        { this._count = val; } });  
Cercle.prototype = {  
    surface: function surface() {  
        return Math.pow(this.rayon, 2) * Math.PI; } };  
Object.defineProperty(Cercle.prototype, "rayon", {  
    get: function() {  
        return this._radius; },  
    set: function(rayon) {  
        if (!Number.isInteger(rayon))  
            throw new Error("rayon est un entier.");  
        this._radius = rayon; } });
```

# Après

```
class Cercle {
    constructor(rayon) {
        this.rayon = rayon;
        Cercle.nbrDeCercles++; };
    static dessiner(cercle, canvas) {
        // Code pour dessiner dans le Canvas };
    static get nbrDeCercles() {
        return !this._count ? 0 : this._count;};
    static set nbrDeCercles(val) {
        this._count = val; };
    surface() { return Math.pow(this.rayon, 2) * Math.PI; };
    get rayon() { return this._radius; };
    set rayon(radius) {
        if (!Number.isInteger(radius))
            throw new Error("Le rayon est un entier.");
        this._radius = radius; }; }
}
```

Pour qu'une sous-classe hérite d'une autre classe on utilisera le mot clé **extends**.



# & more

- Les promesses (ou promise)
  - callback amélioré
- Les templates strings
- Les modules
  - Découpage du code JS en plusieurs fichiers
- ...



```
// ES5
var name = "world";
var myStrin = "Hello " + name;
// => Hello world

// ES6
const newName = `developer`;
const myStrin = `Hello ${newName}`;
// => Hello developer
```