

# JS → ES.NEXT

[\[github\]](#)

## Speaker notes

- Ardoise Shift + F4
- Afficher Ctrl+L
- Utiliser un bloc

```
{  
  // code  
}
```

- Console  
Ctrl+Shift+K
- *Recharger* via le menu

# RÉFÉRENCES

1. [ES6 features.org](#)
2. [lukehoban](#)
3. [ES2016](#) →  
[ES2018](#)

# ECMAScript

- Standard defini par ECMA Intl.
  - LiveScript (Netscape)
  - JavaScript (<sup>TM</sup> Oracle)
  - ES

## Speaker notes

- C'est quoi ce nom ?

# ECMAScript ?

- JS1.5 → ES5
- ES6 → ES2015
- ES2015 →  
ES2017
- ES.Next



## Speaker notes

- C'est quoi ces versions ?
  - Normalisation JS1.5 → ES5
  - Passage à un versionning par année
  - ES.Next = prochaine version








# let / const

- Portée de var : function
-  Portée de bloc avec let
-  Constantes



## Speaker notes




-  {var } visible hors du bloc
  - Comportement avec function
  - Comportement de let
- Variable globale
-  function désormais block-scoped
  -  function test()  
imbriquées

# CLASSES

- Clarifier la notion existante
  - basé sur `__proto__`
  - pas de modif structurelle



## Speaker notes

-  `let obj = {}` avec `nom`, `hello()`
  - syntaxe simplifiée de fonction
  -  String interpolation
  -  property shorthand si var existe déjà `{x, hello}`
- conversion `class Actor`, constructor

```
{
  class Actor {
    constructor(name) {
      this.name = name
    }

    hello() {
      return Hello ${this.name}</code>
    }
  }
  new Actor('Frank Oz').hello()
}
```

# CLASSES

- Propriétés
- Héritage

## Speaker notes

- Héritage simple uniqt. (une seule superclasse)





## Speaker notes

- `get yodaName();`
- `set yodaName(v);`
- `extends, super`
- ⌚ object literals [ `"name" + foo()` ]:

42

```
{
  class Actor {
    constructor(name) {
      this.name = name
    }



    hello() {
      return <code>Hello ${this.name}</code>
    }

    get yodaName() {
      return this.name.split('').reverse().join('')
    }

    set yodaName(v) {
      this.name = v.split('').reverse().join('')
    }
  }
  new Actor('Frank Oz').yodaName
}
```




# MODULES

- $n$  fichiers de script =  $n$  balises `<script>`  

- Visibilité
- Collision de noms (entre librairies)
- Solutions  :
  - Utilisation d'un bundler (webpack...)
  - `(function(){})( )`

## Speaker notes

- Organiser une grande codebase

# MODULES

-  Modules ES
  - `import, export`
- Bien supporté
- Mode strict par défaut


## Speaker notes

- Quelques restrictions  
(./)





## Speaker notes

-  classe Actor + hello dans une fonction + export
- une classe Film avec une liste d'acteurs + hello
- Import entre scripts
  - named import {}, export default
- Import dans le navigateur
  - `<script type=module>`



# PROG. FONCTIONNELLE



# ARROW FUNCTIONS

- Simplifier les déclarations à la volée
- `function(x) {}`  $\rightarrow$  `(x) => ...`





## Speaker notes

- Comportement quasi-identique
-  `Array.map` avec fonction `(x) { }`  
(Acteurs)
  - Version avec `() =>`
  - `filter`, `forEach`
-  trailing commas dans `[]`

# LEXICAL *this*

- Eviter la perte du contexte






## Speaker notes

- Comportement quasi-identique
- `this` = contexte exec. function





## Speaker notes

-  `let obj = {}` avec valeurs, copies, `initCopie` avec `forEach(function())`
  - Contournement avec `self (vm, $ctrl)`
  - Remplacer par `(e) => this.copies.unshift(e)`
-  ASI
-  trailing commas dans `{}`




```
{
  let obj = {
    acteurs: ['Oscar Isaac', 'Mark Hamill', 'Daisy Riley'],
    copies: [],

    initCopies: function () {
      this.valeurs.forEach(function(e) {
        this.copies.unshift(e)
      })
    }
  }
  obj.initCopies()
  obj.copies
}
```

# DÉCOMPOSITION



## Speaker notes

-  tableau `let [a, b, c] = array`
  - Swap deux valeurs `[a, b] = [b, a]`
  - Ignorer `let [a, , c] = array`
  - Par défaut `let [a, b, c = 3] = array`
  - Fail-soft `let [a, b, c = 3] = [0]`
-  object `let { name, age } =`  
`getActor()`
  - `let { n:name, a:age } = getActor()`
-  paramètre de fonction `function ([a, b])`

```
{  
  function test([first, second]) {  
    return first * 10 + second  
  }  
  
  test([1, 2])  
}
```



# FONCTIONS++

-  Paramètres par défaut
- [ ES2017] trailing comma in function args



# REST / SPREAD

- Opérateur . . .
- Autres paramètres (rest)
- Itération (spread)





## Speaker notes

- Rest ...
  - 🖥️ `generique()` réalisateur, producteur, ...acteurs
- Spread ...
  - 🖥️ passer une liste à `generique()`
  - 📦 Utiliser un Set
- Concaténation de listes `[a, b, ...list]`
  - Clonage `[...list]`
  - Marche aussi pour les string, Map, Set, les objets `{...obj}` [🚀 ES2018]
- 🖥️ somme des carrés d'une liste
  - [🚀 ES2016] opérateur

```
{
  function générique(réalisateur, producteur, ...acteurs) {
    return <code>Réalisé par: ${réalisateur}
      Produit par: ${producteur}
      Avec:</code> + acteurs.join(',')
  }

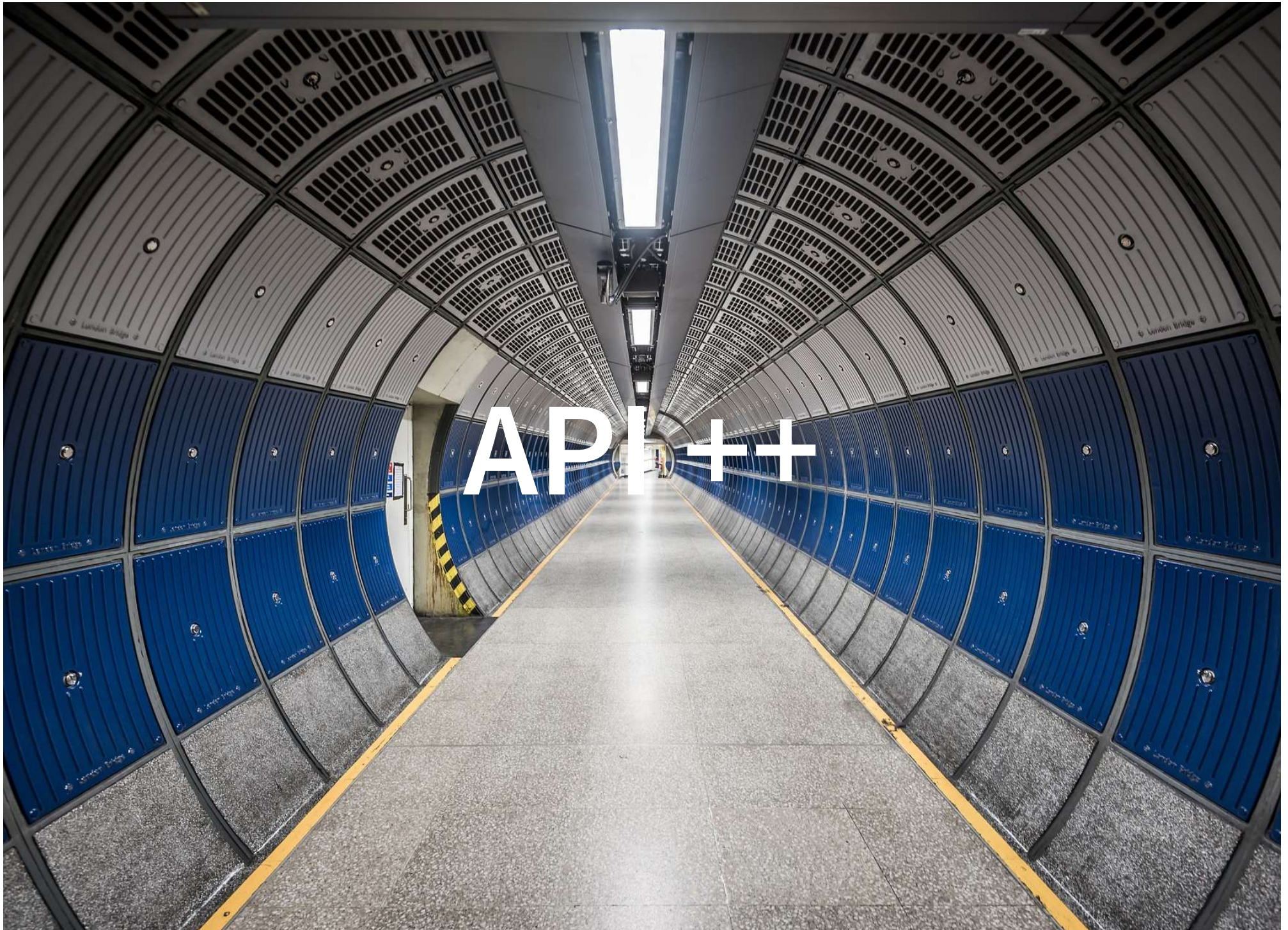
  générique('Irvin Kershner', 'George Lucas', 'Mark Hamill', 'Carrie Fisher', 'Harrison Ford')

  let whoSwho = new Set(['Irvin Kershner', 'George Lucas', 'Mark Hamill', 'Carrie Fisher', 'Harrison Ford', 'George Lucas'])
  générique(...whoSwho)
}

{
  function somme(a, ...tail) {
    let f = x => x ** 2
    if (!a) return 0
    return f(a) + somme(...tail)
  }

  somme(1, 2, 3, 4)
}
```







# PROMISE


- Formalisation des promesses (lib. \$q)
- Prog. asynchrone (vs. monothread)

## Speaker notes

- Navigateur monothread (performances / sécurité)
- Asynchrone = ne pas freezer l'UI



## Speaker notes

-  service avec methode getActors
  - Utiliser setTimeout + return
  - Mise en place de la promesse (juste avec resolve)
  - Utilisation, reject
  - ⌚ Promise.all

```
{  
  let service = {  
    getActors() {  
      return new Promise((resolve, _) => {  
        setTimeout(() => resolve(['Nathalie Portman', 'Carrie Fisher', 'Peter Mayhew']), 5000)  
      })  
    }  
  }  
  
  service.getActors().then(data => console.log(data), err => console.error(err))  
}
```






# FETCH

- Simplifier les requêtes distantes
- Asynchrone
  - Renvoie une promesse
- Bye XHR 🙋



## Speaker notes

-  avec [Swapi](#)
  - avec {method:  
'POST'}

```
{  
  fetch('<a href="https://swapi.co/api/films/&#39">https://swapi.co/api/films/&#39</a>;')  
    .then(response => response.json())  
    .then(data => console.log(data.results.map(f => f.title)))  
}
```

# async/await

- [ ES2017]
- Simplifie le chaînage de promesses
- Mais pas que



## Speaker notes

- then() successifs ou imbriqués
- 📖 avec fetch + response.json

```
{
  async function getFilms() {
    let response = await fetch('<a href="https://swapi.co/api/films/&#39">https://swapi.co/api/films/&#39</a>');
    let data = await response.json()
    return data.results.map(f => f.title)
  }

  getFilms().then(data => console.log(data))
}
```



# ET AUSSI...

- `Object.assign()`
- `String`
  - `{repeat, startsWith, ...}`
- Unicode support
- Generator functions
  - `yield`



# RÉFÉRENCES

1. [Canluse.com](#)
2. [You Dont Know JS](#)









# R.O.T.I.

<https://framaforms.org/roti-js-esnext-1526642510>

