Programmation fonctionnelle en JavaScript

Alvin Berthelot

Version 1.0.1

Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons.

Attribution - Partage dans les Mêmes Conditions 3.0 non transposé.

La licence, ses explications ainsi que les moyens de contribution et réappropriation sont détaillés à la fin.

C’est quoi la programmation fonctionnelle ?

Définition de la programmation fonctionnelle

> La **programmation fonctionnelle est un paradigme de programmation** de type déclaratif qui considère le calcul en tant qu'évaluation de fonctions mathématiques. Comme le changement d'état et la mutation des données ne peuvent pas être représentés par des évaluations de fonctions, la programmation fonctionnelle ne les admet pas, au contraire elle met en avant l'application des fonctions, contrairement au modèle de programmation impérative qui met en avant les changements d'état.
>
> — définition Wikipedia

Concepts sous jacents

Pour comprendre comment la programmation fonctionnelle s'applique, il faut comprendre les concepts sous jacents :

- L'**immuabilité**.
- Les **fonctions pures**.
- La **transparence référentielle**.
- Les **fonctions d'ordre supérieur**.

Immuabilité

> Un objet immuable, en programmation orientée objet et fonctionnelle, est un objet dont l'état ne peut pas être modifié après sa création.
>
> — définition Wikipedia

Parmi les valeurs de JavaScript, les primitifs sont immuables par défaut : les chaînes de caractères, les nombres, les booléens et `undefined`.

```
var tutorial = 'Hello';
tutorial += ' World !';
```

Ici la référence `tutorial` a été changé mais pas la valeur de `Hello`. En mémoire il y a donc 2 valeurs `Hello` et `Hello World!`.

La mutation des objets

À l'inverse les objets peuvent muter.

```
var messageHello = {label: 'Hello', name: 'World !'};
var messageGoodbye = messageHello;
messageGoodbye.label = 'Goodbye';

messageHello === messageGoodbye; // Vrai
messageHello.label; // 'Goodbye'
```

Ici la référence de `messageGoodbye` n'a pas été changé mais sa valeur oui.

Les tableaux étant des objets c'est aussi vrai pour eux.

```
var animals = ['Grumpy', 'Droopy'];
var pets = animals;
animals.push('Nemo');

animals === pets; // Vrai
pets.length; // 3
```

Immuabilité Vs Constante

On confond souvent immuabilité et constante, **l'immuabilité empêche la modification de la valeur** alors que **la constante empêche la modification de la référence**.

```
const tutorial = 'Hello';
tutorial += ' World !'; // Erreur

const message = {label: 'Hello', name: 'World !'};
messageGoodbye.label = 'Goodbye'; // Pas d'erreur
```

Quizz Immuabilité Vs Constante

```
var user1 = {firstname: 'John', age: 33, money: 1200};
const user2 = {firstname: 'Jane', age: 31, money: 800};
var user3 = {firstname: 'Paul', age: 16, money: 400};
var message = 'Hello';

function sayMessageToUser(message, user) {
  if(user.age < 18) {
    message += ', but sorry you are too young';
    console.log(message + ' ' + user.firstname);
  } else {
    user.money -= 200;
    console.log(message + ' ' + user.firstname);
  }
}

sayMessageToUser(message, user1); // ???
console.log(message); // ???
console.log(user1.money); // ???

sayMessageToUser(message, user2); // ???
console.log(message); // ???
console.log(user2.money); // ???

sayMessageToUser(message, user3); // ???
console.log(message); // ???
console.log(user3.money); // ???
```

La mutation est-ce grave docteur ?

Un objet peut très facilement être muté même étant déclaré comme constante.

Si un objet est passé en paramètre d'une fonction, on peut se poser là question de comment la fonction va manipuler cet objet et éventuellement le faire évoluer.

Si un objet qui est passé en paramètre d'une fonction est modifié (au sein de cette fonction), on parlera alors d'une **fonction ayant des "effets de bord"** ou de **fonction impure**.

```
function sayMessageToUser(message, user) {
  if(user.age < 18) {
    message += ', but sorry you are too young';
    console.log(message + ' ' + user.firstname);
  } else {
    user.money -= 200;
    console.log(message + ' ' + user.firstname);
  }
}
```

Les fonctions pures

On appelle **une fonction pure, une fonction dont le résultat ne dépend que des arguments qui lui sont passés** et jamais du contexte dans laquelle elle est invoquée **et qui n'a pas d'effets de bord**, c'est à dire qui ne change pas le contexte dans laquelle elle est invoquée.

```
var coeff = 1;

// Pure
function sum(a, b, c) {
  return a + b + c;
}

// Impure
function multiply(a, b, c) {
  return a * b * c * coeff;
}

// Impure
function increaseCoeff() {
  coeff++;
}
```

L’intérêt des fonctions pures

Utiliser un maximum de fonctions pures apportent de nombreux avantages :

- La lisibilité et la compréhension du code.
- La simplification des tests.
- La transparence référentielle.

La simplification des tests

Comme le résultat d'une fonction pure ne dépend que des arguments qui lui sont passés, celà implique qu'**une invocation avec des arguments identiques retournera toujours le même résultat**.

```
var coeff = 1;

// Pure
function getMax(number1, number2) {
  return Math.max(number1, number2);
}

// Impure
function getRandomArbitrary(min, max) {
  return Math.random() * (max - min) + min;
}
```

Cela permet de **tester facilement toute fonction pure**.

La transparence référentielle

Lorsqu'on indique qu'une fonction pure ne dépend que des arguments qui lui sont passés, **il s'agit des évaluations de ces arguments et non de leurs références**.

> Une expression est référentiellement transparente si elle peut être remplacée par sa valeur sans changer le programme.
>
> — définition Wikipedia

```
function addTwo(number) {
  return number + 2;
}
addTwo(5) === addTwo(addTwo(3)); // Vrai
```

Cela permet de **combiner des fonctions à l'intérieur d'autres fonctions**.

Programmation fonctionnelle avec les tableaux

Les tableaux

En JavaScript, les collections sont représentées par des tableaux.

Rappel
Les tableaux sont des objets particuliers héritant de `Array.prototype` et donc disposant de propriétés et méthodes spécifiques.

Les tableaux possèdent 3 principales familles de méthodes :

- Les méthodes mutateurs
- Les méthodes accesseurs
- Les méthodes itératives

Méthodes mutateurs

Les tableaux disposent d'un jeu de méthodes pour manipuler son contenu, voici les plus intéressantes :

- `push()` : ajoute un élément au tableau (et renvoie sa nouvelle longueur).
- `shift()` : supprime et renvoie le premier élément d'un tableau.
- `sort()` : trie le contenu d'un tableau.

```
var animals = ['Grumpy', 'Droopy', 'Nemo'];
var cat = animals.shift();
console.log(animals); // ['Droopy', 'Nemo']
```

Ces méthodes modifient l'état du tableau.

Méthodes accesseurs

Les tableaux disposent d'un jeu de méthodes pour retourner une représentation de son contenu, voici les plus intéressantes :

- `indexOf()` : retourne le premier index d'un élément.
- `includes()` : détermine si élément est présent.
- `concat()` : concaténe des tableaux.
- `join()` : créée une chaîne de caractères à partir d'un tableau.
- `slice()` : créée une copie d'une portion d'un tableau.

```
var animals = ['Grumpy', 'Droopy', 'Nemo'];
var index = animals.indexOf('Nemo');
console.log(index); // 2
```

Méthodes itératives

Les tableaux disposent d'un jeu de méthodes pour itérer sur ses éléments et les transformer, voici les plus intéressantes :

- `map()` : applique une transformation sur chaque élément.
- `reduce()` : applique une fonction sur un accumulateur et sur chaque élément.
- `filter()` : filtre les éléments d'un tableau selon un prédicat.
- `forEach()` : appelle une fonction sur chaque élément.

```
var animals = ['Grumpy', 'Droopy', 'Poppy'];
animals.forEach(function(animal) {
  console.log(animal);
});
```

Manipulation avec les méthodes mutateur

Il faut faire très attention lorsque l'on manipule les tableaux avec les méthodes mutateur car celles-ci génèrent des effets de bord (sur les tableaux) et c'est ce que l'on souhaite éviter au maximum en programmation fonctionnelle.

C'est pourquoi on utilise d'autres méthodes pour cloner le tableau original pour ensuite pouvoir **manipuler cette nouvelle référence**.

```
var animals = ['Grumpy', 'Droopy', 'Nemo'];

var pets = animals.slice(0);

var cat = pets.shift();

console.log(animals); // ['Grumpy', 'Droopy', 'Nemo']
console.log(pets); // ['Droopy', 'Nemo']
```

Manipulation avec les méthodes itératives

À l'inverse lorsque l'on manipule les tableaux avec les méthodes itératives, celles-ci ne génèrent pas d'effets de bord (sur les tableaux).

C'est pourquoi on affecte systématiquement le résultat à une nouvelle variable pour pouvoir **manipuler cette nouvelle référence**.

```
var animals = ['Grumpy', 'Droopy', 'Nemo'];

var pets = animals.map(animal => 'Hello ' + animal);

console.log(animals); // ['Grumpy', 'Droopy', 'Nemo']
console.log(pets); // ['Hello Grumpy', 'Hello Droopy', 'Hello Nemo']
```

Une fonction pure doit toujours être invoquée avec des paramètres (input) et son retour (output) doit être affecté, sinon il y a comme un problème.

Les fonctions d'ordre supérieur

Les fonctions sont des valeurs, il est ainsi possible de les manipuler comme n'importe quel autre valeur, donc de les passer en paramètre ou en retour d'une autre fonction.

On appelle ainsi **une fonction d'ordre supérieur, une fonction qui prend une autre fonction comme argument et/ou qui retourne une fonction comme résultat**.

```
function sum(a, b, c) {
  return a + b + c;
}
sum(1, 2, 3); // retourne 6 mais n'affiche rien

function logEverything(toEvaluate, a, b, c) {
  var result = toEvaluate(a, b, c);
  console.log(result);
  return result;
}

logEverything(sum, 1, 2, 3); // retourne 6 et l'affiche
```

Combinaison de fonctions

En y regardant de plus près, les méthodes itératives de l'objet `Array` **prennent des fonctions en paramètre**, il s'agit donc de fonctions d'ordre supérieure.

Ce qui nous permet de commencer un premier niveau de combinaison.

```
var users = [
  {firstname: 'John', age: 33, money: 1200},
  {firstname: 'Jane', age: 31, money: 800},
  {firstname: 'Paul', age: 16, money: 400}
];

var adults = users.filter(user => user.age >= 18);
var poors = users.filter(user => user.money < 1000);
```

Ici on vient de s'appuyer sur la méthode `filter()` en la combinant avec d'autres fonctions qui sont indépendantes entre elles.

Combinaison de fonctions chaînées

En y regardant de plus près, on s'aperçoit que les méthodes itératives s'appliquent sur des tableaux et elles vont retourner des tableaux sur lesquels on peut également appliquer des méthodes itératives.

Ce qui nous permet de **combiner les fonctions en les chaînant**.

```
var users = [
  {firstname: 'John', age: 33, money: 1200},
  {firstname: 'Jane', age: 31, money: 800},
  {firstname: 'Paul', age: 16, money: 400}
];

var adultsReceiveSalary = users
  .filter(user => user.age >= 18)
  .map(user => { user.money += 1000; return user; });
```

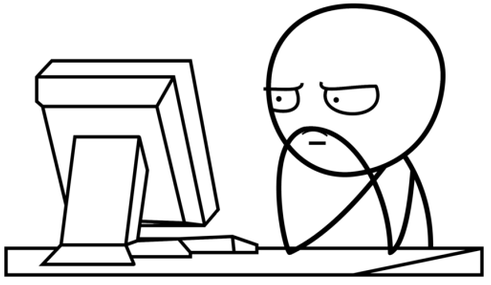
L'ordre d'appel des fonctions a son importance sur le résultat.

La puissance des combinaisons

On vient de voir qu'il était très facile de combiner des fonctions "simples" pour obtenir différents comportements complexes, cela permet de rendre son code plus modulaire. Pour autant, il faut déjà avoir accès à un certain nombre de fonctions "simples".

En JavaScript plusieurs librairies proposent déjà un certain nombre de fonctions génériques :

- Underscore, une librairie proposant un certain nombre de fonctions utiles dans l'approche de la programmation fonctionnelle.
- Lodash, un fork d'Underscore qui a su dépasser son maître en terme de performance.
- Ramda, librairie conçue spécialement pour la programmation fonctionnelle en facilitant le chaînage des fonctions et l'immuabilité des données.



Exercices

On va s'appuyer sur la plateforme NodeSchool.io, pour cela vous devez installer le TP "Functional Javascript".

⇒ Hello World
⇒ Higher Order Functions
⇒ Map
⇒ Filter
⇒ Reduce

Lodash

C’est quoi Lodash ?

Lodash est une librairie de fonctions permettant de manipuler des données et dispose de nombreuses fonctions "utilitaire".

Il s’agit d’un fork d’Underscore, utilisée avec la même notation _.

Pourquoi Lodash ?

Précédemment nous avons vu qu'il était possible de faire de la programmation fonctionnelle avec les tableaux.

Cependant on trouve vite les limites des méthodes natives des tableaux, elles sont trop peu nombreuses, Lodash en propose beaucoup plus.

Les méthodes natives des tableaux ne s'appliquent qu'aux tableaux, Lodash lui permet d'appliquer un certain nombre de fonctions sur différents types, notamment des collections (un mix tableaux et objets).

Il existe deux syntaxes pour manipuler des données.

Utiliser Lodash avec un "wrapper"

On convertit les données à manipuler via un "wrapper" Lodash.

```
var wrappedNames = _(['John', 'Jane', 'Paul']);
var messageNames = wrappedNames.map(user => `Hello ${user}`));

var wrappedUsers = _({adult: 'John', young: 'Paul'});
var messageUsers = wrappedUser.map(user => `Hello ${user}`));
```

Utiliser Lodash en passant les données en paramètre

Il est également possible de passer les données à manipuler en paramètres.

```
_.map(['John', 'Jane', 'Paul'], user => `Hello ${user}`);

_.map({adult: 'John', young: 'Paul'}, user => `Hello ${user}`);
```

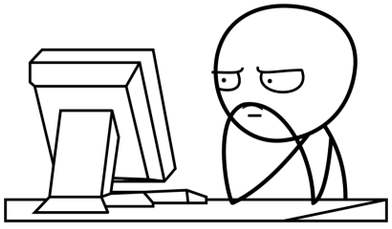
C'est plutôt cette syntaxe qui est utilisée, c'est d'ailleurs celle de la documentation officielle.

Le chaînage de fonctions

Cependant avec la syntaxe de manipulation en passant les données en paramètre, il n'est pas si facile de chaîner les fonctions.

Il est nécessaire de passer par un objet "wrapper" en le constituant avec la fonction `chain` et en le déconstituant avec la fonction `value`.

```
var adultsReceiveSalary = _.chain([
  {firstname: 'John', age: 33, money: 1200},
  {firstname: 'Jane', age: 31, money: 800},
  {firstname: 'Paul', age: 16, money: 400}
])
.filter(user => user.age >= 18)
.map(user => { user.money += 1000; return user; })
.value();
```



Exercices

On va s'appuyer sur la plateforme NodeSchool.io, pour cela vous devez installer le TP "LololoDash".

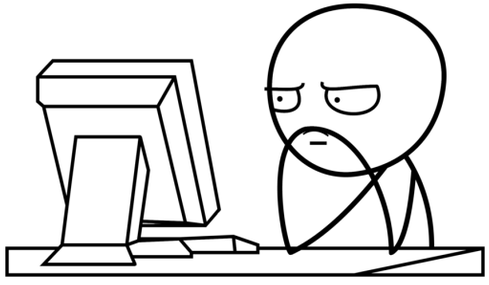
⇒ Getting Started
⇒ Sort Me
⇒ In Every Case
⇒ Everyone Is Min
⇒ Chain Mail
⇒ Count the Comments
⇒ Give Me an Overview
⇒ Analyze
⇒ Start templating

La récursivité

Définition

> Un algorithme récursif est un algorithme qui résout un problème en calculant des solutions d'instances plus petites du même problème.
>
> — définition Wikipedia

```
function factorial(num)
{
  // If the number is less than 0, reject it.
  if (num < 0) { return -1; }
  // If the number is 0, its factorial is 1.
  else if (num === 0) { return 1; }
  // Otherwise, call this recursive procedure again.
  else { return (num * factorial(num - 1)); }
}

console.log(factorial(8)); // 40320
```



Exercices

On va s'appuyer sur la plateforme NodeSchool.io, pour cela vous devez installer le TP "Functional Javascript".

⇒ Basic: Recursion
⇒ Recursion

La curryfication

Définition

> En programmation fonctionnelle, la **curryfication** désigne la **transformation d'une fonction à plusieurs arguments en une fonction à un argument qui retourne une fonction sur le reste des arguments**.
>
> — définition Wikipedia

Imaginons une fonction f avec 3 arguments : a, b et c.

Avec une fonction non currifiée `f(a)` retourne **l'évaluation** de `f(a, undefined, undefined)`. Avec une fonction currifiée `f(a)` retourne une fonction `g(b, c)`.

Cette technique permet de convertir une fonction ayant de multiples paramètres en une séquence de fonctions ayant chacune un unique paramètre.

Implémentation en JavaScript

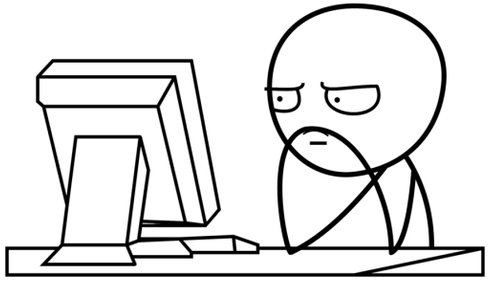
Si la curryfication est native dans certains langages de programmation (comme Haskell), ce n'est pas le cas en JavaScript où il faut implémenter ce comportement.

```
var hello = function(firstname, lastname) {
  return `Hello ${firstname} ${lastname}`
};

hello('John'); // 'Hello John undefined'

function curry2(fn) {
  return function(firstArg) {
    return function(secondArg) {
      return fn(firstArg, secondArg);
    };
  };
}

var helloCurried = curry2(hello);

var helloFirstName = helloCurried('John');
helloFirstName('Doe'); // 'Hello John Doe'

helloCurried('Jane')('Fonda'); // 'Hello Jane Fonda'
```

Curryfication avec librairies

Les librairies Lodash et Ramda proposent des fonctions facilitant la mise en place de la curryfication.

```
var hello = function(firstname, lastname) {
  return `Hello ${firstname} ${lastname}`
};

// Lodash
var helloCuriedWithLodash = _.curry(hello);
```



Exercices

On va s'appuyer sur la plateforme NodeSchool.io, pour cela vous devez installer le TP "Currying in JavaScript".

⇒ IDENTITY
⇒ BINARY
⇒ DELAY INVOCATION
⇒ CURRY FUNCTION

Licence

Pour toutes questions, réclamations ou remarques, merci d'envoyer un message à alvin.berthelot@webyousoon.com.

Explications licence CC BY-SA 3.0

Cette licence permet aux autres de remixer, arranger, et adapter votre œuvre, même à des fins commerciales, tant qu'on vous accorde le mérite en citant votre nom et qu'on diffuse les nouvelles créations selon des conditions identiques.

Cette licence est souvent comparée aux licences de logiciels libres, "open source" ou "copyleft".

Toutes les nouvelles œuvres basées sur les vôtres auront la même licence, et toute œuvre dérivée pourra être utilisée même à des fins commerciales.

C'est la licence utilisée par Wikipédia ; elle est recommandée pour des œuvres qui pourraient bénéficier de l'incorporation de contenu depuis Wikipédia et d'autres projets sous licence similaire.

Contribution et réappropriation

Ce fichier PDF est généré avec Asciidoctor à partir d'un dépôt Git se trouvant sous GitHub.

https://github.com/alvinberthelot/slides-functional-js

Cela signifie que vous n'avez pas besoin de vous battre avec un fichier binaire (le PDF) pour **contribuer**, **vous réapproprier le contenu** ou **modifier le thème** de présentation.

Contribution

Vous voulez **contribuer au contenu** car :

- Il y a une erreur (ça arrive à tout le monde), de typographie, de compréhension, ou tout autre chose.
- Vous souhaitez apporter une précision.

Il vous suffit de contribuer au projet via Git par le moyen d'une "pull request" sur le dépôt Git.

Réappropriation

N'oubliez pas les conditions de la licence.

Vous voulez vous **réapproprier le contenu** car :

- Vous souhaitez donner un style différent.
- Vous souhaitez enlever/ajouter/modifier des sections dans votre contexte.

Il vous suffit de "forker" le dépôt Git et d'y apporter vos propres modifications, puis de générer par vous même le nouveau PDF.