# Programmation fonctionnelle en JavaScript: 20 OU 2 ?







### Langages pratiqués

- 1. Java
- 2. JavaScript (TypeScript, CoffeeScript)
- 3. Kotlin, Scala

Notions dans

Python, SML, Racket, Rust, Swift, Go, ...



### Roadmap

- I. Langages fonctionnels
- II. Programmation fonctionnelle en JavaScript Part I
- III. Entract
- IV. Programmation fonctionnelle en JavaScript Part II
- V. Remaque sur la performance
- VI. Conclusion



# LANGAGES FONCTIONNELS



Fooling around with alternating current (AC) is just a waste of time. Nobody will use it, ever. — Edison, 1889

\*

There is no reason anyone would want a computer in their home. — Ken Olson, 1977

11

I predict the Internet will soon go spectacularly supernova and in



- C est un langage fonctionel
- JavaScript est un langage fonctionel
- Java est un langage 00
- JQuery est une monade

#### Paradigmes

- programmation impérative
- programmation orientée objet
- programmation fonctionnelle

•

On peut adopter donc un style de programmation fonctionnelle avec la plupart des langages. Les carcactéristiques des langages peuvent rendre cela plus ou moins facile (voir obligatoire)



C'est quoi un langage fonctionnel?



Il n'y a gun langage fonctionnel : le  $\chi$ -calcul



developpeur fonctionnel => Devel

(Haskell, normaux, humour, paresseux, lambda)

### Static vs Dynamic



<del>r rogramation fonctionnelle 7 - rypage statique</del>

Dynamique

Lisp (1958)

Racket (1994), Clojure (2007), ...

Statique

ML (1973)



PROGRAMMATION FONCTIONNELLE EN JAVASCRIPT - PART



```
function
   return a * b;
typeof mult; // "function"
Anonymous
var mult = function (a, b) {
   return a * b;
ES2015
```



```
var sum = 0;
[1, 2, 3, 4, 5]
    .forEach(elt ⇒ sum += elt);
console.log({sum});
```

danger, c'est un nid à bugs.

=> Éviter les fonctions qui n'ont pas de paramètres, ou retournent void

```
const sum = [1, 2, 3, 4, 5]
    reduce((acc. elt) ⇒ acc += elt):
use1S @ilaborie
```



const

Object.freeze

avec ImmutableJS



### High Order function

map, reduce, filter, sort

```
// f: X \Rightarrow Y

// g: Y \Rightarrow Z

// compose: (Y \Rightarrow Z, X \Rightarrow Y) \Rightarrow X \Rightarrow Z

const compose = (g, f) \Rightarrow x \Rightarrow g(f(x))
```



```
const factorial3 = (n, acc = 1) ⇒
    (n ≤ 1) ? acc : factorial3(n - 1, acc * );

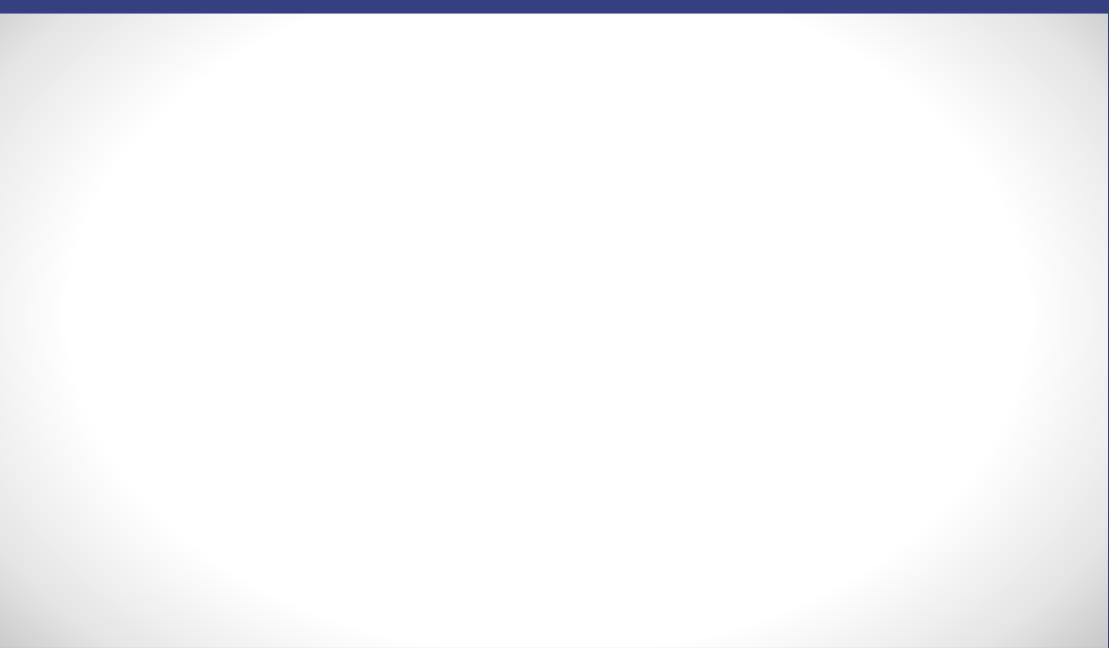
Support de la tail-rec)

### 		Éviter les recusions entre plusieurs fonctions:
```

```
const a = n ⇒ {
    if (n > 42) {
        return b(n - 42);
    }
    return 2;
}
```



```
speakers
                     .filter(speaker ⇒ speaker.xp > 10 &
                                                       speaker.some(lang ⇒ lang ≡ 'JavaScript'))
speakers
                     .filter(speaker \Rightarrow speaker.xp > 10) // is experimental 
                     .filter(speaker \Rightarrow speaker.some(lang \Rightarrow lang \equiv '
const isExperimented = speaker ⇒ speaker.xp > 10;
const isLoveJS = speaker ⇒ speaker.loves.some(lang =
speakers
                     .filter(isExperimented)
                      .filter(isLoveJS)
```





### ENTRACT



#### PandaRoux



By Adrian Pingstone (Photographed by Adrian Pingstone) [Public domain], >> via Wikimedia Commons

Photo by Lance Anderson on Unsplash

Ailurus fulgens

Habitat: the Himalayan forests of Nepal and China.



PROGRAMMATION FONCTIONNELLE EN JAVASCRIPT - PART II



### Jargon



A functor is a collection of X that can apply a function f : X → Y over itself to create a collection of Y.

(Array with map)

Reduction

Function: (X, X) -> X

Combinators

Composition



same result when called in sequence with the same arguments.

```
f(x, y, z) = g(x)(y)(z)
const mult = (a: number, b: number) \Rightarrow a * b;
// idea: identity = mult(1, _)
const identity = (a: number) \Rightarrow mult(1, a);
// better
const multCurry = (a: number) ⇒ {
    return (b: number) \Rightarrow a * b;
```

```
const fibonacci = function(n) {
    switch (n) {
        case 1 : return 1;
        case 2 : return 1;
        default:
            return fibonacci(n-2) + fibonacci(n-1);
    }
}
```

```
datatype race Whole of int
| Frac of int*int
```

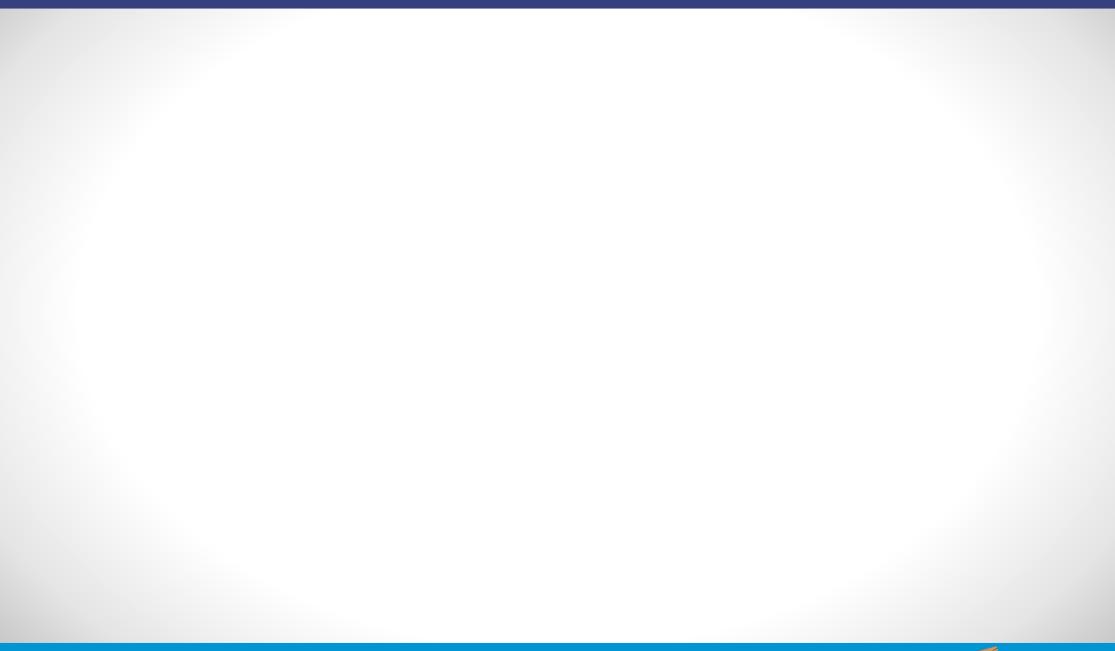
Pattern matching

Pas encore, mais

```
let getLength = vector ⇒ match (vector) {
    { x, y, z }: Math.sqrt(x ** 2 + y ** 2 + z ** 2)
    { x, y }: Math.sqrt(x ** 2 + y ** 2),
    [...]: vector.length,
    else: {
        throw new Error("Unknown vector type");
    }
}
```

### Pattern Matching



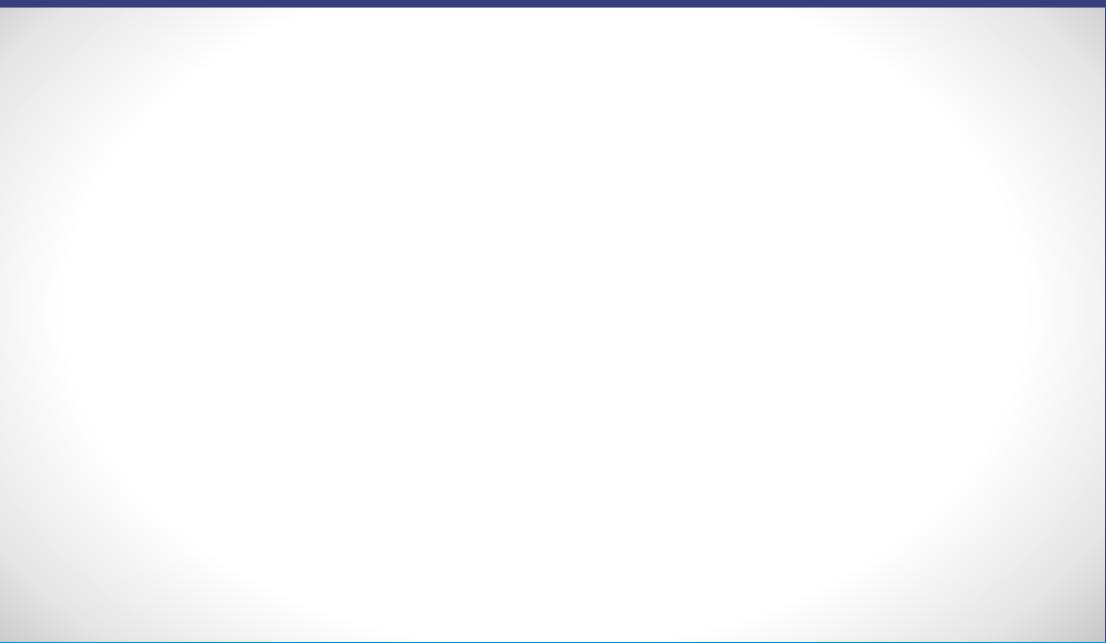




```
Some(1)
   .map { it + 1 }
   .flatMap { Some("4" + it) }
   .map { it }
```



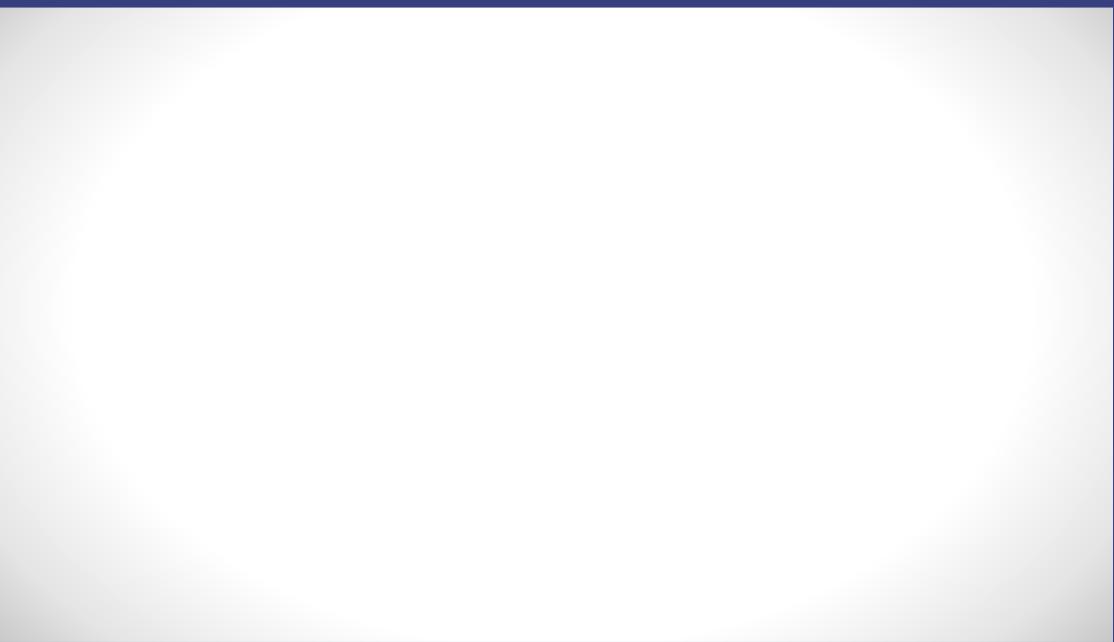


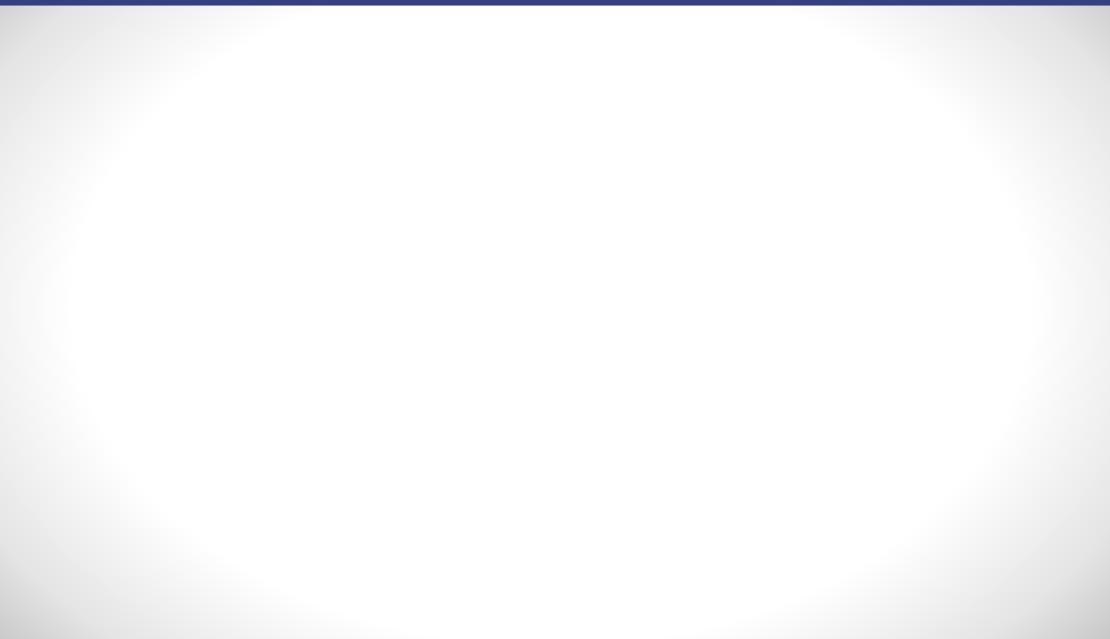


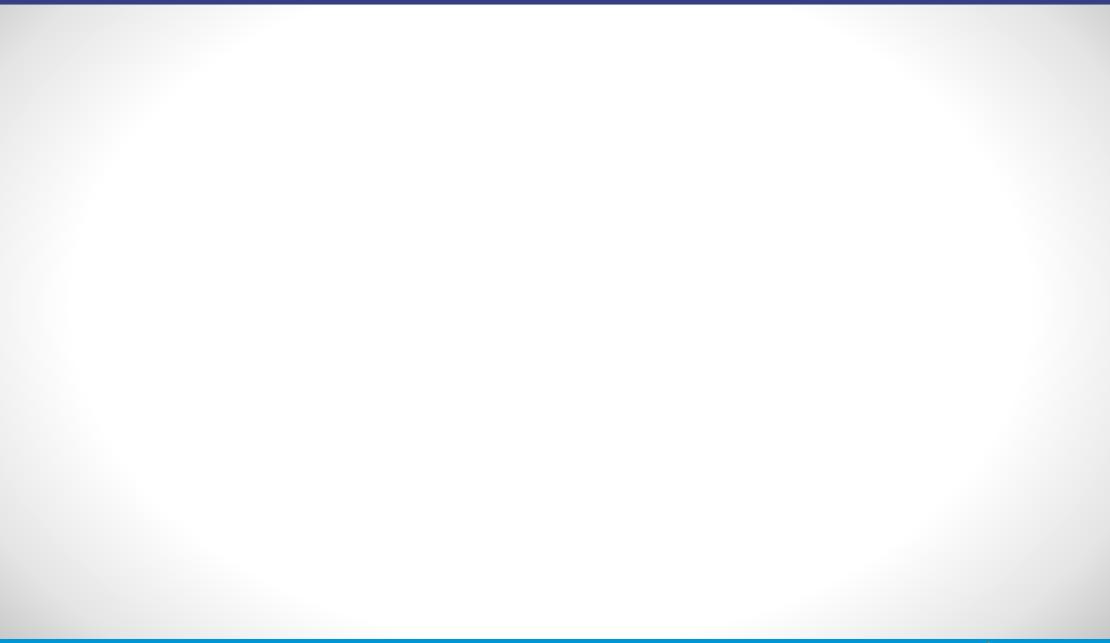


## REMAQUE SUR LA PERFORMANCE









### CONCLUSION



- les bases sont présentes.
- écosystème dans ce domaine, plutôt à la mode

Mais encore insuffisant à nom gout:

- flatMap 🐿
- un lazy Array (comme les **Stream** Java)







Program designers have a tendency to think of the users as idiots who need to be controlled. They should rather think of their program as a servant, whose master, the user, should be able to control it. If designers and programmers think about the apparent mental qualities that their programs will have, they'll create programs that are easier and pleasanter — more humane — to deal with.

-- John McCarthy "The Little Thoughts of Thinking Machines", Psychology Today, December 1983, pp. 46–49. Reprinted in Formalizing Common Sense: Papers By John McCarthy, 1990, ISBN 0893915351



La valeur principale d'un programme est dans ce qu'il réalise, pas dans son style.

Mais en temps qu'artisan développeur, j'accorde de l'importance au style.



expressivité souplesse ecosysteme évolu dans le bon sens



- Plus facile à tester
- moins de bugs
- plus simple
- plus évolutif \*



slides

• Fantasy Land Specification (aka "Algebraic JavaScript Specification")



Questions?

