

Programmation fonctionnelle en JavaScript



Philippe Charrière

<https://github.com/k33g/q/issues>



GitHub



@k33g

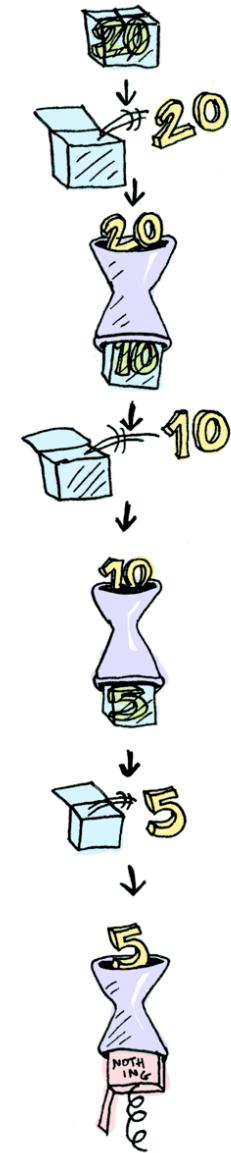
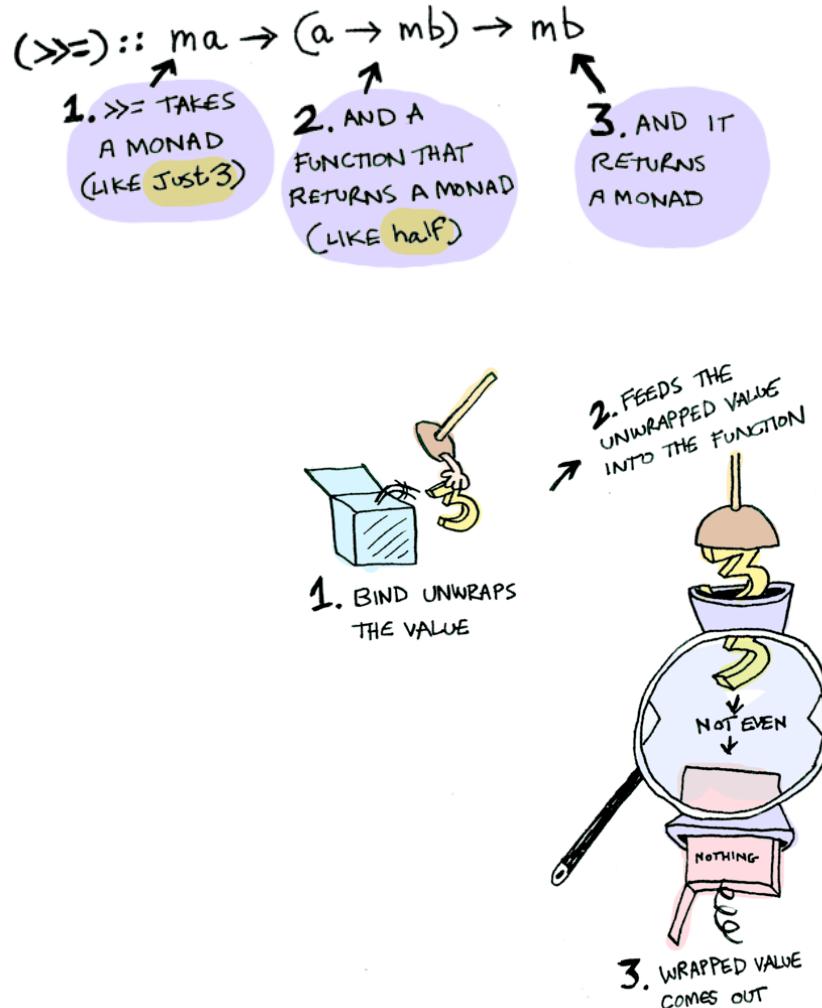


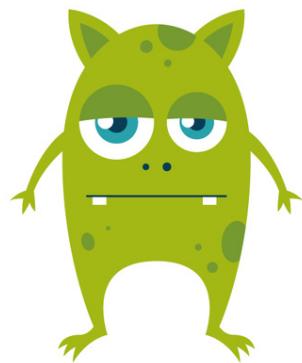
@k33g_org

mix-IT



Pourquoi ce talk? ... Pour qui?





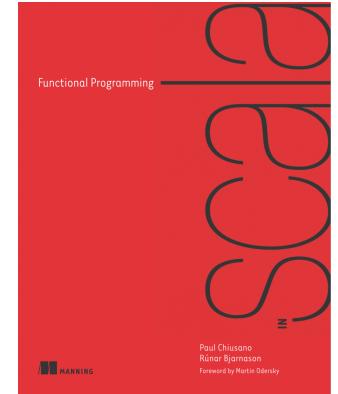
Précautions d'usage

- > vocabulaire(s)
- > implémentations incomplètes
- > échanges de points de vue:

<https://github.com/k33g/voxxed.lu.functional.js/issues>

Programmation fonctionnelle?

Définition



- > Coder avec uniquement des fonctions pures
- > = fonctions sans “side effect”
 - > toujours retourner le même résultat
 - > ne pas modifier de variable
 - > ne pas propager d’exception ou s’arrêter lors d’une erreur
 - > pas de print, pas de input
 - > pas de lecture / écriture de fichier
 - > ...



développeurs fonctionnels

Nous allons voir

- > 2, 3 trucs utiles
- > Container
- > Functor
- > Monad
- > Maybe
- > Either
- > Applicative Functor
- > Validation

Quelques bases utiles

“currying is a way of constructing functions that allows partial application of a function’s arguments”

```
let addition = function(a,b) {  
    return a+b;  
};  
  
// currying addition  
let add = function(a) {  
    return function(b) {  
        return a + b;  
    }  
};
```

```
add(40)(2)
```

```
incrementBy1 = add(1)
incrementBy2 = add(2)
```

```
incrementBy1(41)
```

```
let arr1 = [100, 200, 300, 400, 500];
```

```
arr1.map((x)=>{ return x + 1 })
```

```
arr1.map(incrementBy1)
```

“Function composition is the act of pipelining the result of one function, to the input of another, creating an entirely new function.”

```
compose2 = (f, g) => {
    return function(x) {
        return f(g(x));
    }
};
```

```
compose3 = (f, g, h) => {
    return function(x) {
        return f(g(h(x)));
    }
};
```

Mancy - REPL(0)

```
✓ addOne = n => n + 1
▶ () {}

✓ multiplyByFive = n => n * 5
▶ () {}

✓ a1m5 = compose2(multiplyByFive, addOne)
▶ function (x) {}

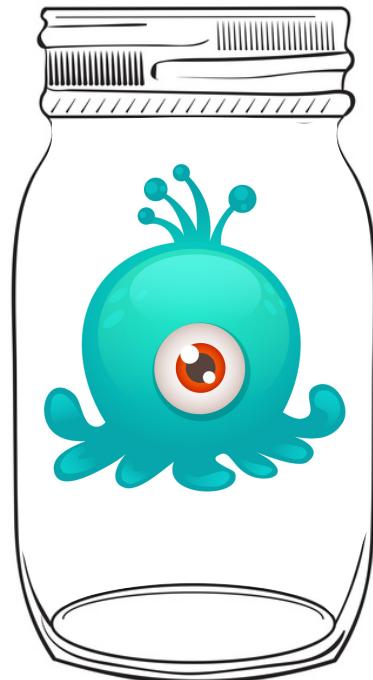
✓ a1m5(4)
25
.....
✓ m5a1 = compose2(addOne, multiplyByFive)
▶ function (x) {}

✓ m5a1(4)
21
.....
> 1
```



REPL for fun 🎨

Container



```
class Container {  
    constructor(x) {  
        const value = x;  
        Object.defineProperty(this, "value", { get: () => value })  
    }  
  
    static of(x) {  
        return new Container(x);  
    }  
}
```

```
bob = Container.of("Bob Morane");
▼ Container {} #
  value: "Bob Morane"
  __proto__: ► Container {}

bob.value
"Bob Morane"

bob.value = 'John Doe'
"John Doe"

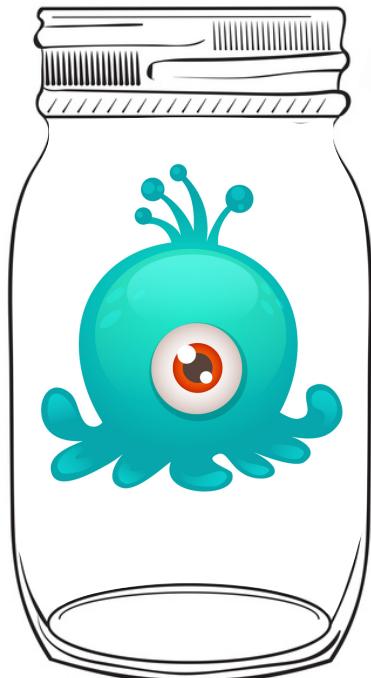
bob
▼ Container {} #
  value: "Bob Morane"
  __proto__: ► Container {}

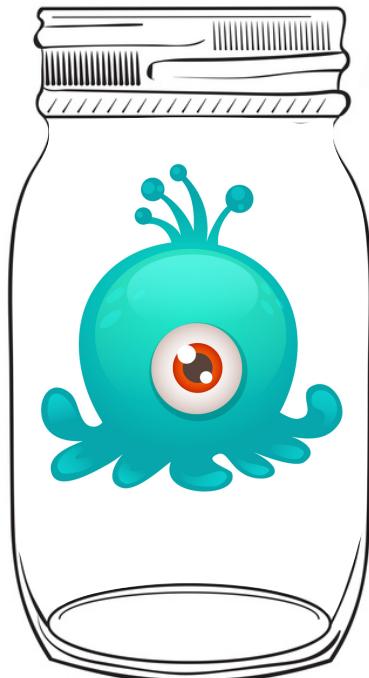
> 1
```



REPL for fun 🎉

Functor











```
class Functor {  
    constructor(x) {  
        const value = x;  
        Object.defineProperty(this, "value", { get: () => value })  
    }  
  
    static of(x) {  
        return new Functor(x);  
    }  
  
    map (fn) {  
        return Functor.of(fn(this.value));  
    }  
}
```

I'm a
Functor!



`map(function)`

Mancy - REPL(1) ✖️ ⏪ ⌂

```
✓ panda = Functor.of('💀')
  ▼ Functor {} #
    value: "💀"
    __proto__: ► Functor {}

✓ addLapinouBuddy = (me) => me + '🐰'
  ► (me) {}

✓ buddies = panda.map(addLapinouBuddy)
  ▼ Functor {} #
    value: "💀🐰"
    __proto__: ► Functor {}

✓ addCatBuddy = (me) => me + '🐱'
  ► (me) {}

✓ buddies.map(addCatBuddy)
  ▼ Functor {} #
    value: "💀🐰🐱"
    __proto__: ► Functor {}

✓ panda
  ▼ Functor {} #
    value: "💀"
    __proto__: ► Functor {}

✓ buddies
  ▼ Functor {} #
    value: "💀🐰"
    __proto__: ► Functor {}

> 1
```



REPL for fun 🐾

✖️ ⏪ ⌂

“un peu” plus sérieusement

```
addOne = (value) => value + 1;  
multiplyBy5 = (value) => value * 5;  
divideByThree = (value) => value / 3;  
  
a = Functor.of(23.2);  
  
b = a  
  .map(addOne)  
  .map(addOne)  
  .map(multiplyBy5)  
  .map(divideByThree);
```

Monad





```
☰ Mancy - REPL(1) ✎ C

▼ panda = Functor.of('💀')
  ▼ Functor {} #
    value: "💀"
    __proto__: ► Functor {}

▼ addTigrouBuddy = (me) => Functor.of(me + '🐯')
  ► (me) {}

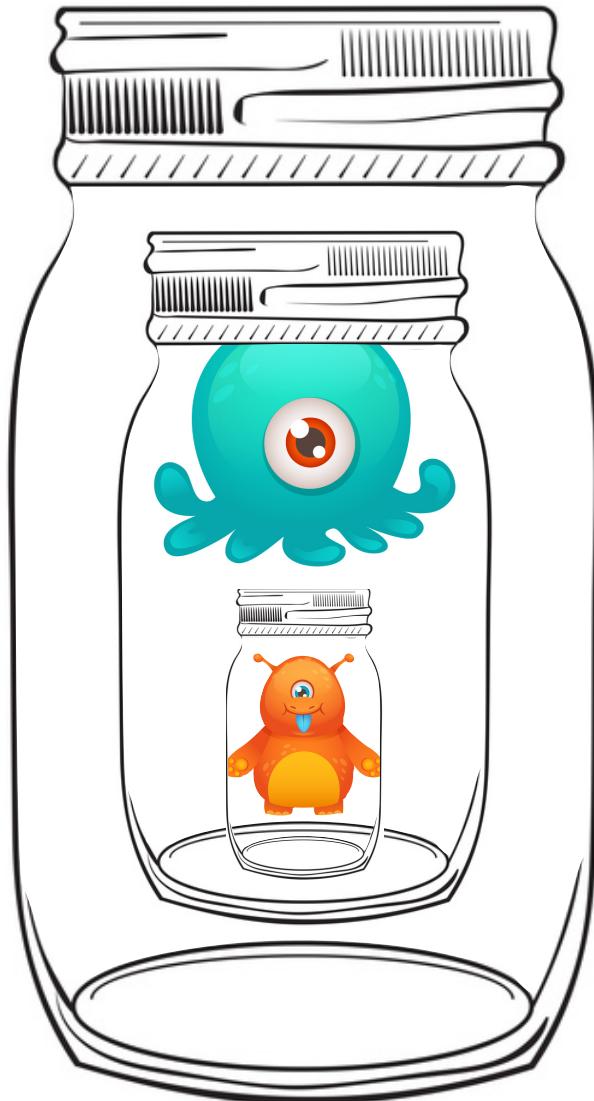
▼ buddies = panda.map(addTigrouBuddy)
  ▼ Functor {} #
    value: ▼ Functor {} #
      value: "💀🐯"
      __proto__: ► Functor {}
    __proto__: ► Functor {}

▼ buddies.value.value
  "💀🐯"

> 1
```



REPL for fun 🦸



```
class Monad {  
    constructor(x) {  
        const value = x;  
        Object.defineProperty(this, "value", { get: () => value })  
    }  
  
    static of(x) {  
        return new Monad(x);  
    }  
  
    map (fn) {  
        return Monad.of(fn(this.value));  
    }  
  
    fmap (fn) { // flatMap, bind  
        return fn(this.value);  
    }  
}
```

```
☰ Mancy - REPL(1) ✎ C

▼ panda = Monad.of('💀')
  ▼ Monad {} #
    value: "💀"
    __proto__: ► Monad {}

▼ addTigrouBuddy = (me) => Monad.of(me + '🐯')
  ► (me) {}

▼ buddies = panda.fmap(addTigrouBuddy)
  ▼ Monad {} #
    value: "💀🐯"
    __proto__: ► Monad {}

▼ fullOfBuddies = panda
  .fmap(addTigrouBuddy)
  .fmap(addTigrouBuddy)
  .fmap(addTigrouBuddy)
  .fmap(addTigrouBuddy)

  ▼ Monad {} #
    value: "💀🐯🐯🐯🐯"
    __proto__: ► Monad {}

> 1
```

REPL for fun 🐾

I'm a
Monad!

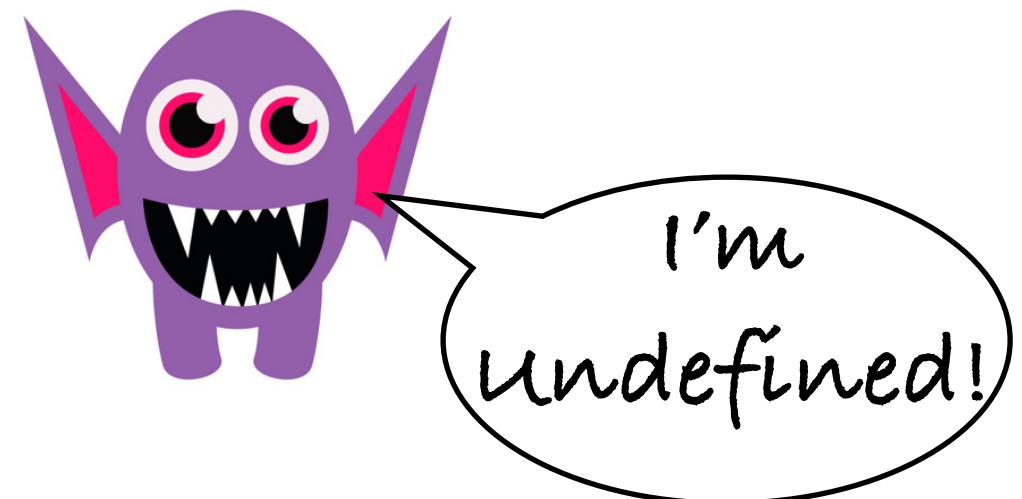


fmap (function)

map (function)

Maybe

Pain points



```
bob = {  
  id:'bob',  
  address:{  
    email:'bob@github.com',  
    country:'US'  
  }  
};  
  
john = {  
  id:'john',  
  address:{  
    country:'US'  
  }  
};  
  
jane = {  
  id:'jane'  
};  
  
buddies = [bob, john, jane];
```

```
getMailById = (id, buddies) => {  
  let email = buddies  
    .find(buddy => buddy.id == id)  
    .address.email;  
  
  if(email === null || email === undefined) {  
    return "no email"  
  }  
  return email  
};
```



```
✓ buddies
▶ Array[3] ✖️ ⌂ C

✓ getMailById('bob', buddies)
bob@github.com ↗ ✖️ ⌂ C

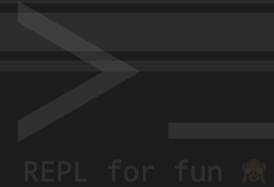
✓ getMailById('john', buddies)
"no email" ✖️ ⌂ C

✓ getMailById('jane', buddies)
▶ TypeError: Cannot read property 'email' of undefined ✖️ ⌂ C

✓ getMailById('sam', buddies)
▶ TypeError: Cannot read property 'address' of undefined ✖️ ⌂ C
```

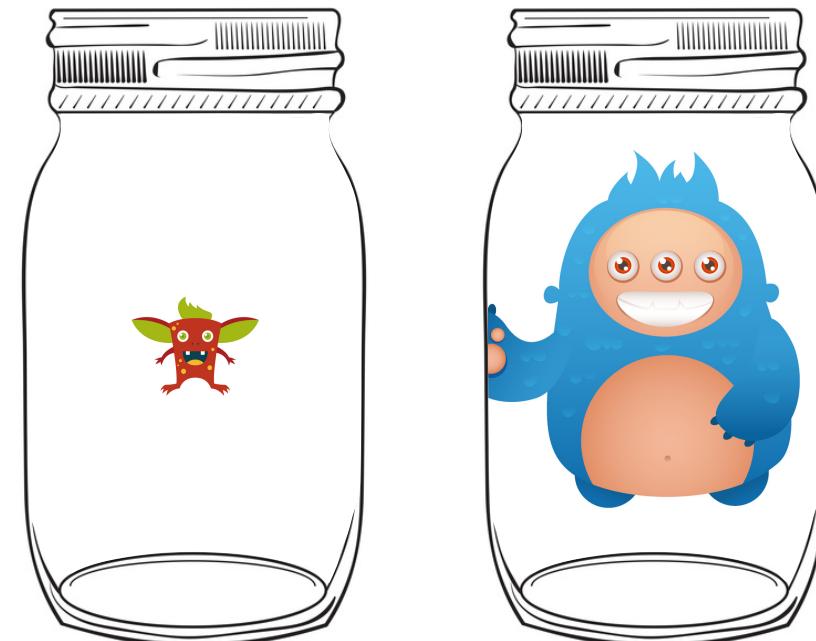


1



REPL for fun 🎉

Maybe None Some



```
class Maybe {  
    static Some(x) { return new Some(x); }  
  
    static None() { return new None(); }  
  
    static fromNullable(x) {  
        return (x !== null && x !== undefined)  
            ? Maybe.Some(x)  
            : Maybe.None();  
    }  
  
    static of(x) { return Maybe.Some(x); }  
}
```

```
class None {
    constructor() {
        const value = null;
        Object.defineProperty(this, "value", { get: () => value })
    }

    map() { return this; }

    fmap (fn) { return this.value; }

    getOrElse(value) { return value; }

    isNone() { return true; }

    isSome() { return false; }
}
```

```
class Some {
  constructor(x) {
    const value = x;
    Object.defineProperty(this, "value", { get: () => value })
  }

  map(fn) {
    let res = fn(this.value);
    if(res == null || x == undefined) { return new None(); }
    return new Some(res);
  }

  fmap (fn) { return fn(this.value); }

  getOrElse() { return this.value; }

  isNone() { return false; }

  isSome() { return true; }
}
```

“functional” refactoring

```
bob = {  
  id:'bob',  
  address:{  
    email:'bob@github.com',  
    country:'US'  
  }  
};  
  
john = {  
  id:'john',  
  address:{  
    country:'US'  
  }  
};  
  
jane = {  
  id:'jane'  
};  
  
buddies = [bob, john, jane];
```

```
getMailById = (id, buddies) => {  
  return Maybe.fromNullable(  
    buddies.find(buddy => buddy.id == id)  
    .map(buddy => buddy.address)  
    .map(address => address.email)  
    .getOrDefault('no email!'))  
};
```

```
✓ buddies
▶ Array[3]

✓ getMailById = (id, buddies) => {
  return Maybe.fromNullable(buddies.find(buddy => buddy.id == id))
    .map(buddy => buddy.address) // return address of buddy
    .map(address => address.email) // return email of address
    .getOrDefault('no email!')
};

▶ (id, buddies) {}

✓ getMailById('bob', buddies)
bob@github.com ↗

✓ getMailById('john', buddies)
"no email!"

✓ getMailById('jane', buddies)
"no email!"

✓ getMailById('sam', buddies)
"no email!"
```



Maybe

null, “expliqué” par le type: **None()**
pas d’information sur l’erreur

I'm a
maybe!

J'ai 2 sous types

None
Some



fmap (function)

map (function)

Either

Either Left Right



```
class Either {  
    constructor(x) {  
        const value = x;  
        Object.defineProperty(this, "value", { get: () => value })  
    }  
  
    static Left(x) { return new Left(x); }  
  
    static Right(x) { return new Right(x); }  
  
    static fromNullable(x) {  
        return (x !== null && x !== undefined)  
            ? Either.Right(x)  
            : Either.Left();  
    }  
  
    static of(x) { return Either.Right(x); }  
}
```

```
class Left {  
    constructor(err) {  
        const value = err;  
        Object.defineProperty(this, "value", { get: () => err })  
    }  
  
    map() { return this; }  
  
    fmap (fn) { return this; }  
  
    getOrElse(value) {return value; }  
  
    orElse(fn) { return fn(this.value); }  
  
    cata(leftFn, rightFn) { return leftFn(this.value); }  
}
```

```
class Right {  
    constructor(x) {  
        const value = x;  
        Object.defineProperty(this, "value", { get: () => value })  
    }  
  
    map(fn) { return new Right(fn(this.value)); }  
  
    fmap (fn) {  
        let res = fn(this.value);  
        if(res == null || res == undefined) {return new Left(res);}  
        return new Right(res);  
    }  
  
    getOrElse() { return this.value; }  
    orElse() { return this; }  
  
    cata(leftFn, rightFn) { return rightFn(this.value); }  
}
```

utilisation

```
function getMonthName(mo) {  
    mo = mo-1; // Adjust month number for array index (1=Jan, 12=Dec)  
    var months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul",  
        "Aug", "Sep", "Oct", "Nov", "Dec"];  
    if (months[mo] !== undefined) {  
        return months[mo];  
    } else {  
        throw new UserException("Invalid Month Number");  
    }  
}
```

```
function getDayName(da) {  
    da = da-1; // Adjust day number for array index  
    var days = ["Mo", "Tu", "We", "Th", "Fr", "Sa", "Su"];  
    if (days[da] !== undefined) {  
        return days[da];  
    } else {  
        throw new UserException("Invalid Day Number");  
    }  
}
```



getMonthName(12)

"Dec"

getMonthName(13)

▼ UserException: Invalid Month Number

```
at UserException.ExtendableException (repl:6:7)
at UserException (repl:3:1)
at getMonthName (mancy-repl:8:13)
at mancy-repl:1:1
at ContextifyScript ctxt.runInContext (/Applications/Mancy-darwin-x64/Mancy.app/Contents/Resources/app/common/ReplContext.js ↗ :203:18)
at Object.runInContext (/Applications/Mancy-darwin-x64/Mancy.app/Contents/Resources/app/common/ReplCommon.js ↗ :427:23)
at ReplActiveInput.transpileAndExecute (/Applications/Mancy-darwin-
x64/Mancy.app/Contents/Resources/app/components/ReplActiveInput.js ↗ :678:32)
at /Applications/Mancy-darwin-x64/Mancy.app/Contents/Resources/app/components/ReplActiveInput.js ↗ :843:20
```

getDayName(42)

▼ UserException: Invalid Day Number

```
at UserException.ExtendableException (repl:6:7)
at UserException (repl:3:1)
at getDayName (mancy-repl:18:13)
at mancy-repl:1:1
at ContextifyScript ctxt.runInContext (/Applications/Mancy-darwin-x64/Mancy.app/Contents/Resources/app/common/ReplContext.js ↗ :203:18)
at Object.runInContext (/Applications/Mancy-darwin-x64/Mancy.app/Contents/Resources/app/common/ReplCommon.js ↗ :427:23)
at ReplActiveInput.transpileAndExecute (/Applications/Mancy-darwin-
x64/Mancy.app/Contents/Resources/app/components/ReplActiveInput.js ↗ :678:32)
at /Applications/Mancy-darwin-x64/Mancy.app/Contents/Resources/app/components/ReplActiveInput.js ↗ :843:20
```



REPL for fun

on “cache” les exceptions

```
giveMeMonthName = (monthNumber) => {
    try {
        return Either.Right(getMonthName(monthNumber));
    } catch (err) {
        return Either.Left(err);
    }
};
```

```
giveMeDayName = (dayNumber) => {
    try {
        return Either.Right(getDayName(dayNumber));
    } catch (err) {
        return Either.Left(err);
    }
};
```

```
✓ giveMeMonthName(12)
  ▼ Right {} #
    value: "Dec"
    __proto__: ► Right {}

✓ giveMeMonthName(42)
  ▼ Left {} #
    value: ► UserException: Invalid Month Number
    __proto__: ► Left {}

✓ giveMeDayName(1)
  ▼ Right {} #
    value: "Mo"
    __proto__: ► Right {}

✓ giveMeDayName(42)
  ▼ Left {} #
    value: ► UserException: Invalid Day Number
    __proto__: ► Left {}
```



> 1

catamorphisme

```
● ● ● Mancy - REPL(0) ✖ ⏪ ⏴ C

▼ giveMeMonthName(12).cata(
  leftFn= (err) => {
    return JSON.stringify(err);
  },
  rightFn= (value) => {
    return `Yes! Month name is ${value}`;
  }
)
"Yes! Month name is Dec"

▼ giveMeMonthName(42).cata(
  leftFn= (err) => {
    return JSON.stringify(err);
  },
  rightFn= (value) => {
    return `Yes! Month name is ${value}`;
  }
)
"{"name": "UserException"}"

> 1
```

The logo for 'REPL for fun' features a large, stylized greater-than sign (>) positioned above a horizontal bar. Below the bar, the text 'REPL for fun' is written in a small, sans-serif font, with a small icon of a person's head next to the word 'fun'.

mais...

```
checkMonthAndDay = (monthNumber, dayNumber) => {
    try {
        let month = getMonthName(monthNumber);
        let day = getDayName(dayNumber);
        return Either.Right({month, day});
    } catch (err) {
        return Either.Left(err);
    }
}
```

```
checkMonthAndDay = (monthNumber,dayNumber) => {  
  ► (monthNumber,dayNumber) {}  
  
  ↘ checkMonthAndDay(1,42)  
    ▼ Left {} #  
      value: ► UserException: Invalid Day Number  
      __proto__: ► Left {}  
  
  ↘ checkMonthAndDay(42,5)  
    ▼ Left {} #  
      value: ► UserException: Invalid Month Number  
      __proto__: ► Left {}  
  
  ↘ checkMonthAndDay(42,42)  
    ▼ Left {} #  
      value: ► UserException: Invalid Month Number  
      __proto__: ► Left {}  
  
> 1
```



Mancy - REPL(0)

✖️ ⚡ C

✖️ ⚡ C

✖️ ⚡ C

✖️ ⚡ C

On ne trace pas toutes les erreurs

I'm an
Either!

J'ai 2 sous types

Left
Either



fmap(function)

map(function)

cata(
left function,
right function)

Validation

Applicative Functor ???

c'est un  avec
une méthode **ap()**

ap(functor)



map(function)

Méthode ap()

“ap is a function that can apply the function contents of one functor to the value contents of another”

ap(functor)



map(function)

```
class ApplicativeFunctor {
  constructor(x) {
    const value = x;
    Object.defineProperty(this, "value", { get: () => value })
  }
  static of(x) {
    return new ApplicativeFunctor(x);
  }

  map (fn) {
    return ApplicativeFunctor.of(fn(this.value));
  }

  ap(otherContainer) { // has to be a functor
    return otherContainer.map(this.value)
  }
}
```

```
● ● ● Mancy - REPL(0) x - C  
▼ addTwo = (n) => n + 2  
► (n) {}  
x ap1 = ApplicativeFunctor.of(addTwo)  
▼ ApplicativeFunctor {} #  
  value: ▼ (n) {} #  
    length: 1  
    name: ""  
    __proto__: ► Function {}  
    (n) => n + 2  
  __proto__: ► ApplicativeFunctor {}  
x ap2 = ApplicativeFunctor.of(40)  
▼ ApplicativeFunctor {} #  
  value: 40  
  __proto__: ► ApplicativeFunctor {}  
x ap3 = ap1.ap(ap2)  
▼ ApplicativeFunctor {} #  
  value: 42  
  __proto__: ► ApplicativeFunctor {}  
> 1
```



```
☰ Mancy - REPL(0) ✎ C

▼ add = a => b => a + b
  ▼ () {} #
    length: 1
    name: ""
    __proto__: ► Function {}
    a => b => a + b

▼ add(40)(2)
  42
  .....

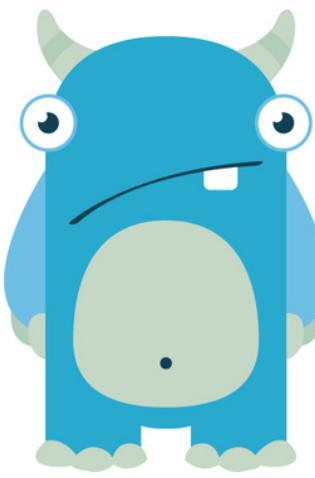
▼ ApplicativeFunctor.of(2)
  .map(add)
  .ap(ApplicativeFunctor.of(40))

▼ ApplicativeFunctor {} #
  value: 42
  __proto__: ► ApplicativeFunctor {}

> 1
```



REPL for fun 🎉



validation

Fail Success



```
class Validation {
  constructor(x) {
    const value = x;
    Object.defineProperty(this, "value", { get: () => value })
  }

  static Success(x) { return new Success(x); }

  static Fail(x) { return new Fail(x); }

  static of(x) { return Validation.Success(x); }
}
```

```
class Success {
  constructor(err) {
    const value = err;
    Object.defineProperty(this, "value", { get: () => err })
  }

  map(fn) { return new Success(fn(this.value)); }

  isSuccess() { return true; }
  isFail() { return false; }

  ap(otherContainer) { // has to be at least a functor
    return otherContainer instanceof Fail
      ? otherContainer
      : otherContainer.map(this.value)
  }

  cata(failureFn, successFn) { return successFn(this.value); }
}
```

```
class Fail {
    constructor(x) {
        const value = x;
        Object.defineProperty(this, "value", { get: () => value })
    }

    map() { return this; }

    isSuccess() { return false; }
    isFail() { return true; }

    ap(otherContainer) { // has to be a functor
        return otherContainer instanceof Fail
            ? new Fail(this.value.concat(otherContainer.value))
            : this
    }

    cata(failureFn, successFn) { return failureFn(this.value); }
}
```

exemple(s)

```
checkMonth = (monthNumber) => {
  try {
    let month = getMonthName(monthNumber);
    return Validation.Success(month)
  } catch (err) {
    return Validation.Fail([err])
  }
};
```

```
checkDay = (dayNumber) => {
  try {
    let day = getDayName(dayNumber);
    return Validation.Success(day)
  } catch (err) {
    return Validation.Fail([err])
  }
};
```



```
✓ checkDay(42)
▼ Fail {} #
  value: ▼Array[1] #
    0: ► UserException: Invalid Day Number
    length: 1
    __proto__: ► Array {}
  __proto__: ► Fail {}

✓ checkDay(42).ap(checkMonth(42)).ap(checkMonth(42))
▼ Fail {} #
  value: ▼Array[3] #
    0: ► UserException: Invalid Day Number
    1: ► UserException: Invalid Month Number
    2: ► UserException: Invalid Month Number
    length: 3
    __proto__: ► Array {}
  __proto__: ► Fail {}

✓ checkDay(42).ap(checkMonth(12)).ap(checkMonth(42))
▼ Fail {} #
  value: ▼Array[2] #
    0: ► UserException: Invalid Day Number
    1: ► UserException: Invalid Month Number
    length: 2
    __proto__: ► Array {}
  __proto__: ► Fail {}
```



```
checkMonthOnData = (data) => {
  try {
    let month = getMonthName(data.month);
    return Validation.Success(month)
  } catch (err) {
    return Validation.Fail([err])
  }
};

checkDayOnData = (data) => {
  try {
    let day = getDayName(data.day);
    return Validation.Success(day)
  } catch (err) {
    return Validation.Fail([err])
  }
};
```

```
checkMonthAndDay = (data) => Validation.of(
  day => {
    console.log('dayName:', day)
    return month => {
      console.log('monthName:', month)
      return `==> ${day}:${month}`
    };
  }).ap(checkDayOnData(data)).ap(checkMonthOnData(data));
```

```
● ● ● Mancy - REPL(0) ✖ ⏪ C  
✓ checkMonthAndDay = (data) => Validation.of(  
  day => {  
    console.log('dayName:', day)  
    return month => {  
      console.log('monthName:', month)  
      return `==> ${day}: ${month}`  
    };  
}).ap(checkDayOnData(data)).ap(checkMonthOnData(data))  
▶ (data) {}  
  
✓ checkMonthAndDay({day:2,month:12}) ✖ ⏪ C  
  ▼ Success {} #  
    value: "==> Tu:Dec"  
    __proto__: ▶ Success {}  
  
✓ checkMonthAndDay({day:42,month:42}) ✖ ⏪ C  
  ▼ Fail {} #  
    value: ▼ Array[2] #  
      0: ▶ UserException: Invalid Day Number  
      1: ▶ UserException: Invalid Month Number  
      length: 2  
      __proto__: ▶ Array {}  
    __proto__: ▶ Fail {}  
  > 1
```



catamorphisme

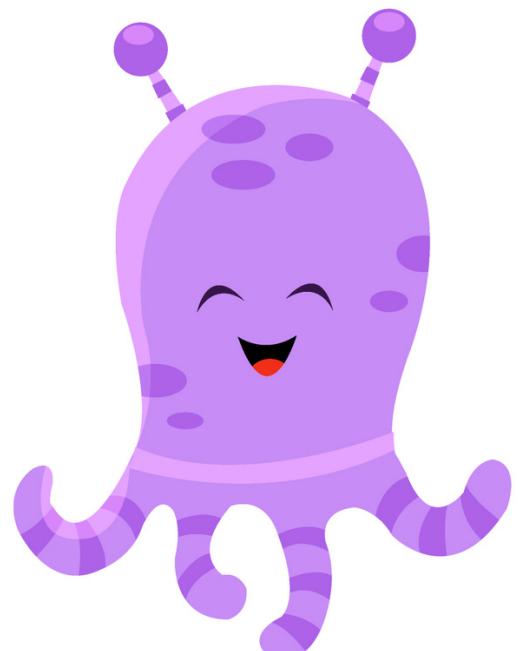
```
✓ checkMonthAndDay = (data) => Validation.of(  
  day => {  
    console.log('dayName:', day)  
    return month => {  
      console.log('monthName:', month)  
      return `==> ${day}:${month}`  
    };  
}).ap(checkDayOnData(data)).ap(checkMonthOnData(data))  
▶ (data) {}  
  
✓ checkMonthAndDay({day:42,month:42}).cata(  
  failureFn = (errors) => errors.join('|'),  
  successFn = (res) => res  
)  
"UserException: Invalid Day Number|UserException: Invalid Month Number"  
  
✓ checkMonthAndDay({day:2,month:12}).cata(  
  failureFn = (errors) => errors.join('|'),  
  successFn = (res) => res  
)  
"==> Tu:Dec"
```



I'm a validation!

J'ai 2 sous types

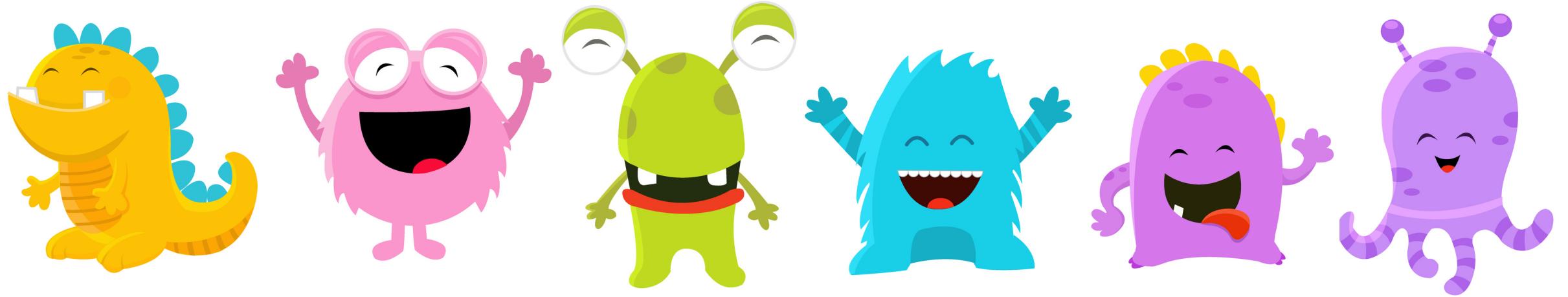
Fail
Success



ap(validation)

map(function)

cata(
failureFn function,
successFn function
)



Paysage Fonctionnel côté JS

Quelques libs

- > Monet.js: <https://cwmyers.github.io/monet.js/>
- > Ramda.js: <http://ramdajs.com/0.21.0/index.html>
- > Underscore: <http://underscorejs.org/>
- > Lodash: <https://lodash.com/>
- > Folktale: <http://folktalejs.org/>
- > Fantasy Land: <https://github.com/fantasyland>

Ressources

Lectures

Mostly Adequate Guide (Brian Lonsdorf): <https://www.gitbook.com/book/drboolean/mostly-adequate-guide/details>

Functional Programming in Java (Pierre-Yves Saumont): <https://www.manning.com/books/functional-programming-in-java>

Functional Programming in JavaScript (Luis Atencio): <https://www.manning.com/books/functional-programming-in-javascript>

Functional Programming in Scala (Paul Chiusano and Rúnar Bjarnason): <https://www.manning.com/books/functional-programming-in-scala>

Le code source de Golo:

<https://github.com/eclipse/golo-lang/blob/master/src/main/golo/errors.golo>

<https://github.com/eclipse/golo-lang/blob/master/src/main/java/gololang/error/Result.java>

Talks

Gérer les erreurs avec l'aide du système de types de Scala !
(David Sferruzza):

<https://www.youtube.com/watch?v=TwJQKrZ23Vs>

TDD, comme dans Type-Directed Development
(Clément Delafargue):

<https://www.youtube.com/watch?v=XhcgCF0xXRs>

Remerciements

Merci



- > Loïc Descotte [@loic_d](https://twitter.com/loic_d)
- > Julien Ponge [@jponge](https://twitter.com/jponge)
- > Thierry Chantier [@titimoby](https://twitter.com/titimoby)

- > [@voxxed_lu](https://twitter.com/voxxed_lu)

<https://github.com/k33g/voxxed.lu.functional.js>

Questions

[https://github.com/k33g/
voxxed.lu.functional.js/issues](https://github.com/k33g/voxxed.lu.functional.js/issues)