LES MONADES

CA ME CASSE LES GONADES

KAIZEN
SOLUTIONS

LES GONADES

(les illustrations sont NSFW)

KAIZEN SOLUTIONS

LES MONADES

« une monade est juste un monoïde dans la catégorie dans endofoncteurs »

KAIZEN SOLUTIONS

MERCI

JULIEN LENORMAND

Ingénieur informatique

KAIZEN SOLUTIONS

KAIZEN SOLUTIONS

26 rue Général Mouton-Duvernet
69003 Lyon

www.kaizen-solutions.net
--
contact@kaizen-solutions.net

DÉFINITION ≠ EXPLICATION ≠ INTUITION

KAIZEN SOLUTIONS

LE CONTEXTE

- EDF en 2019
- 2 façons de faire la gestion d'erreur en Python :

```
result = something_that_may_fail()
if result is None:
    ... # traitement d'erreur
else:
    ... # on poursuit les opérations
```

```
try:
    result = something_that_may_raise()
except FileNotFoundError:
    ... # traitement d'erreur
else:
    ... # on poursuit les opérations
```

KAIZEN SOLUTIONS

LA DÉCOUVERTE

JUNIOR

```python
def _read_file(filepath):
    # may raise a FileNotFoundError (and others ...)
    with open(filepath, "rt") as file:
        return file.read(-1)

def safe_read_file(filepath):
    try:
        file_content = _read_file(filepath)
    except FileNotFoundError:
        return Optional.empty()
    else:
        return Optional.of(file_content)
```

```python
filepath = ...
file_content = safe_read_file(filepath)
if file_content.is_empty():
    # traitement d'erreur
    ...
else:
    # on poursuit les opérations avec
    file_content.value
```

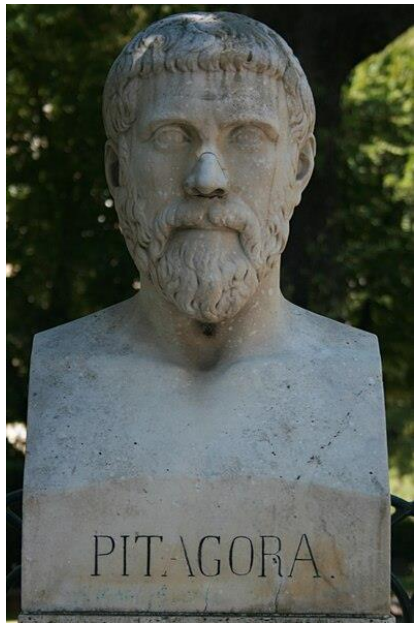
KAIZEN SOLUTIONS

RÉSULTAT

- La nouvelle méthode ressemble beaucoup à juste avoir des *None*
- Maintenant il y a 3 façons de faire la gestion des erreurs
- On a des erreurs avec les *Optional*
- Mes collègues ne comprennent pas
- Et moi non plus

KAIZEN SOLUTIONS

LES MONADES, HISTORIQUEMENT

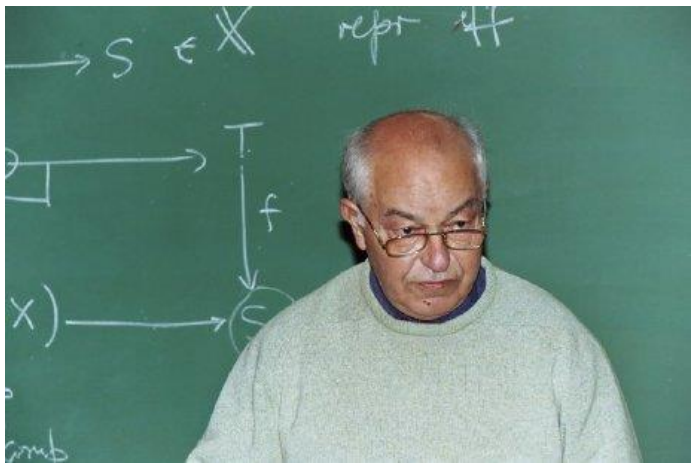


$$V \times V \xrightarrow{M} \mathbb{R}$$
$(v, w) \mapsto v' M w$
$M_{ij} = M(b_i, b_j)$
$M \in V^* \otimes V^*$
$M = M_{st} \beta^s \otimes \beta^t$

$$V \times V^* \xrightarrow{M} \mathbb{R}$$
$(v, f) \mapsto v' M f'$
$M_i{}^j = M(b_i, \beta^j)$
$M \in V^* \otimes V$
$M = M_s{}^t \beta^s \otimes b_t$
$M_s{}^t = M_{su} g^{ut}$
MG^{-1}

$$V^* \times V \xrightarrow{M} \mathbb{R}$$
$(f, w) \mapsto f M w$
$M^i{}_j = M(\beta^i, b_j)$
$M \in V \otimes V^*$
$M = M^s{}_t b_s \otimes \beta^t$
$M^s{}_t = g^{su} M_{ut}$
$G^{-1} M$

$$V^* \times V^* \xrightarrow{M} \mathbb{R}$$
$(f, g) \mapsto f M g'$
$M^{ij} = M(\beta^i, \beta^j)$
$M \in V \otimes V$
$M = M^{st} b_s \otimes b_t$
$M^{st} = g^{su} M_{uv} g^{vt}$
$G^{-1} M G^{-1}$



Haskell

KAIZEN SOLUTIONS

HASKELL

- Recherche / Production
- Strictement typé
- Inférence de type
- Purement fonctionnel (immutabilité et transparence référentielle)
- Evaluation paresseuse
- Monadique (IO)
- Type classes
- Curry

KAIZEN SOLUTIONS

COMMENT EXPLIQUER LES MONADES ?

- *The monad curse*
- Variables :
  - Background mathématique
  - Connaissance de Haskell
  - Connaissance de la prog fonctionnelle
  - Objectif
- Festival des tutoriels et métaphores
- Une histoire de types (« typage »)

KAIZEN SOLUTIONS

C'EST QUOI ?

Ça dépend pour qui :

- Les mathématiciens qui font de l'informatique
- Le langage Haskell (son compilateur)
- Les chercheurs qui travaillent sur Haskell
- Les devs qui débutent en Haskell
  (ou autres langages fonctionnels)
- Les devs qui maîtrisent en Haskell
  (ou autres langages fonctionnels)
- Les devs en d'autres langages

KAIZEN SOLUTIONS

POUR QUOI FAIRE ?

Exemples :

- Option/Maybe
- Result/Either
- List
- State
- Reader
- Writer
- IO
- Exceptions
- Continuations
- Probability
- …

KAIZEN SOLUTIONS

C'EST QUOI, SUPERFICIELLEMENT ?

- Une interface
- 3 règles
- Optionellement : Du sucre syntaxique

```
T = typing.TypeVar("T")
U = typing.TypeVar("U")


class Monadic(typing.Generic[T], metaclass=abc.ABCMeta):
    def __init__(self, value: T) -> None:
        raise NotImplementedError

    @abc.abstractmethod
    def bind(self, f: typing.Callable[[T], U]) -> "Monad[U]":
        """
        Bind operator for the monadic type.
        Should follow the 3 laws :
        - left identity
        - right identity
        - associativity
        """
        raise NotImplementedError
```

KAIZEN SOLUTIONS

C'EST QUOI, PROFONDÉMENT ?

- Un vocabulaire précis (sans consensus entre langages/libs)
- Des définitions remplies de symboles mathématiques
- Un cheminement
- Une forme :
  - gestion des *effects*
  - séquencement ou composition des instructions (*semicolon*)
  - *control flow*
  - dépendance des calculs, …

KAIZEN SOLUTIONS

CA SERT VRAIMENT ?

- En Haskell : oui
- Ailleurs : ça dépend
  - Du langage : Java, Rust, TypeScript, …
  - Du contexte socio-technique
- Et sans même le savoir
  - Java *Streams*
  - TypeScript
  - Rust *Optional* et *Either*
  - C# LINQ

KAIZEN SOLUTIONS

EN CONCLUSION

- Monade = différents concepts
- Monade = différentes perspectives
- Monade = différents contextes
- Monade = 1 forme + 3 lois
- Monade = différente organisation du code
- Monade = difficile à maîtriser

KAIZEN SOLUTIONS

POUR ALLER PLUS LOIN

- Jerf : Functors and Monads for people who have read too many "tutorials"
- Jerf, à propos de confondre nom et adjectif
- De l'impossibilité de transmettre une illumination
- Haskell, Scala, F#, Elixir, OCaml, Idris, Clojure, Elm, PureScript, … Rust, Swift, TypeScript, …
- Concepts fonctionnels : pureness, lazyness, HKTs, union and product types, linear types, existential types, type classes et type constructors, …
- Oslash : une des nombreuses librairies pour avoir des monades en Python
- Pourquoi « fantasy-land »

KAIZEN SOLUTIONS

CRÉDITS IMAGES

- Photo de burritos par Max Griss sur Unsplash
- Photo de collombe sur Wikimedia
- Photo de Guido Van Rossum sur Wikimedia
- Buste de Pythagore sur Wikimedia
- Tableau de Leibniz sur Wikimedia
- Logo de Hacker news sur ycombinator.com
- Logo de erlang sur erlang.org
- Meme de la productivité en Haskell par Tobias Hermann
- Algèbre par Juan Marquez sur Math Fandom
- Image de Cadeau sur LearnYouAHaskell
- Photo de Jean Bénabou par le CNRS
- Photo de Jeremy “Jerf” Bowers sur Github

KAIZEN SOLUTIONS

MERCI

JULIEN LENORMAND

Ingénieur informatique

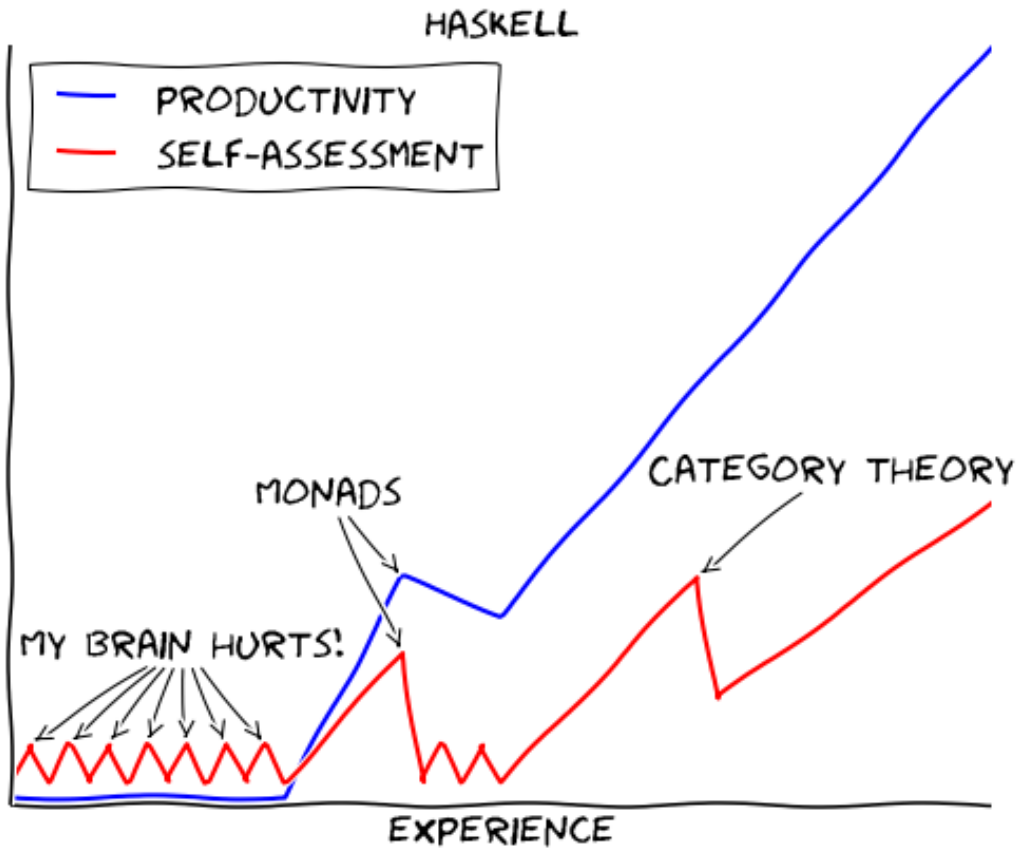
KAIZEN SOLUTIONS

KAIZEN SOLUTIONS

26 rue Général Mouton-Duvernet
69003 Lyon

www.kaizen-solutions.net
--
contact@kaizen-solutions.net

MEME



Hitler reacts to functional programming

KAIZEN SOLUTIONS

HOMMAGE

Il y a deux réponses à cette question, comme à toutes les questions : celle du poète et celle du savant. Laquelle veux-tu en premier ?

-- Pierre Bottero

KAIZEN SOLUTIONS

ANTISÈCHE : CATEGORY THEORY

Functor : fmap respectant identité et composition

```
fmap :: (a -> b) -> f a -> f b
```

Ap functor : ap respectant functor + homomorphisme et interchange

```
(<*>) :: f (a -> b) -> f a -> f b
```

Monoid : mappend respectant ap functor + identité gauche/droite et associativité

```
mappend :: a -> a -> a
```

Monads : monoid + endofunctor

```
(>>=) :: m a -> (a -> m b) -> m b
```

Functor = mappable

Monad = flat-mappable

On suppose tous les functors/applicative/monads comme étant de la catégorie des types (avec certaines limitations informatiques ignorées par rapport au formalisme mathématique)

KAIZEN SOLUTIONS