TDO - S01E03

DEVOPS

mercredi 10 novembre @ Prism

Merci

- au Prism de nous accueillir
- à la CCISM d'avoir créé le Prism
- à vous d'être venu

Cette présentation

- est un fichier Markdown
- versionné avec git
- transformé avec Deckset
- disponible sur github: source et PDF

Buts

- démystifier la programmation fonctionnelle (on en a tous fait au moins une fois, sans le savoir)
- la rendre intelligible par un néophyte (le jargon, les monades, etc...)
- donner envie d'en faire

Sommaire

- les paradigmes de programmation
- la programmation fonctionnelle
- pourquoi
- comment
- un peu d'exotisme

Disclaimer

Certains mots/noms peuvent être inconnus, voire obscurs, mais les concepts ou pratiques qu'ils désignent vous sont certainement familier. Le but est de se doter d'un vocabulaire commun.

Paradigme de programmation ?

Un paradigme de programmation est un style fondamental de programmation informatique qui traite de la manière dont les solutions aux problèmes doivent être formulées dans un langage de programmation.

Différences entre paradigmes et langages

- le paradigme revient à «comment concevoir la solution ?»
- le langage est un outil qui facilite l'utilisation d'un paradigme
- il peut être possible, mais pas toujours, de faire du fonctionnel dans un langage impératif

Les paradigmes principaux

- Wikipedia recense plus de 35 paradigmes 😅
- D'habitude, on commence avec la programmation impérative (Basic, C, PHP, Cobol)
- Puis on enchaîne sur la programmation orientée objet
- Et on peut vivre toute sa vie sans savoir qu'on fait de la programmation déclarative

Le paradigme fonctionnel (1/2)

- est un paradigme déclaratif
- qui considère le calcul en tant qu'évaluation de fonctions mathématiques.
- qui fait peur car largement associé à des mathématiques rébarbatives

Le paradigme fonctionnel (2/2)

- considére les fonctions comme des types valides
- ne permet pas la mutation de variables
- empêche l'utilisation de variables globales
- et n'attire pas les foules

Pourquoi faire du fonctionnel

- encapsulation du code -> plus facile à refactorer
- pas d'effets de bord -> moins de bugs
- pas d'effets de bord -> plus facilement parallélisable

Comment faire du fonctionnel

- l'idéal est de prendre un langage fonctionnel (Haskell est devenu la référence)
- sinon, un langage qui permet de manipuler des fonctions en tant que valeurs (ruby, swift ou kotlin ou autre...)
- faire des fonctions aussi petites que possibles
  - non, encore plus petites que ça

Exemple non fonctionnel

```
def decorate(str)
{
  var upper     = String.uppercase(str)
  var withMarks = upper + " !!!"
  var withName  = "Bidule dit: " + withMarks

  return withName
}

decorate("hello") # "Bidule dit: HELLO !!!"
```

Exemple fonctionnel

```
def upper(str)     { return String.uppercase(str) }
def withMarks(str) { return str + " !!!" }
def withName(str)  { return "Bidule dit: " + str }

def decorate(str)  { return withName ( withMarks ( upper (str) ) ) }
def decorate2(str) { return upper ( withMarks ( withName (str) ) ) }
def decorate3(str) { return withMarks ( upper (str) ) )  }

decorate("hello")  # "Bidule dit: HELLO !!!"
decorate2("hello") # "BIDULE DIT: HELLO !!!"
decorate3("hello") # "HELLO !!!"
```

Factorielle impérative

```
def fact(n)
{
  let product = 1;
  for (let i = 1; i <= n; i += 1)
  {
    product *= n;
  }
  return product;
}
```

Factorielle fonctionnelle (et récursive)

```
function fact(n)
{
  if ( n == 1) return 1
  else         return n * fact( n - 1 )
}
```

Un peu d'exotisme

- théorie des types (ou des catégories)
- curryfication
- functor
- monoïde
- monade

C'est quoi le typage ?

- on parle parfois de langage faiblement typé ou fortement typé
- il s'agit de la capacité du langage à réfléchir en fonction du type de vos valeurs ou instructions
- l'intérêt est de déporter tout un ensemble de problèmes au compilateur