

Programmation Fonctionnelle

Jean-Luc Falcone

HEPIA - 2013

Définitions

- La programmation fonctionnelle:
 - ~~Programmer avec des fonctions~~
- Langage fonctionel:
 - ~~Langage avec des fonctions~~

Transparence référentielle

Une expression est référentiellement transparente si on peut remplacer chacune de ses occurrences avec le résultat de son évaluation sans changer le fonctionnement d'un programme.

Exemple (C/java/...)

//Référentiellement transparente

```
double x = PI / 2;
```

```
double y = sqrt( sin(x)*sin(x) + cos(x)*cos(x) );
```

```
int i = 0;
```

//Référentiellement opaque

```
i = 3;
```

```
int j = ++i;
```

Transparence référentielle (exemples en Scala)

```
val now = currentTime()
```

```
val xs = Array( 0, 0, 0 )  
xs(1) = 1
```

```
val xml = XML.fromFile( "hello.xml" )  
val html = format( xml )  
save( html, "hello.html" )
```

Fonction pures

Une fonction pure est une fonction référentiellement transparente.

Exemple (python)

```
#Fonction pure
def isEmpty( lst ):
    return len(lst) == 0

#Fonction impure
emptyNum=0
def countIfEmpty( lst ):
    if isEmpty(lst):
        emptyNum += 1
    return emptyNum
```

Fonctions Pures (exemple en scala)

```
def randomNoise( x: Double ) =  
  x + rng.nextDouble()/100  
  
def query( db: DataBase, sql: SQL ): Result =  
  db.execute( sql )  
  
def sum( is: Array[Int] ): Int = {  
  var i = 0  
  var sum = 0  
  while( i < is.size ) {  
    sum += is(i)  
    i += 1  
  }  
  sum  
}
```

Définitions

- Programmation fonctionnelle:

Style de programmation basé sur l'utilisation d'expression réf. transparentes et de fonctions pures.

- Langage fonctionnel:

Langage contraignant le style fonctionnel.

Attention

Scala n'est pas un langage fonctionnel (selon cette définition) mais facilite l'utilisation du style fonctionnel.

Avantages

- Pas d'effets de bords
- Composabilité
- Toujours *thread-safe*
- L'ordre de l'évaluation des arguments n'a pas d'importance
- Possibilité d'utiliser un cache
- Facilite l'analyse du code

Désavantages

- Pas d'IO (effets de bord)
 - **Peut** être plus lent (p.e. copie conservative)
 - Nécessite des structures de données appropriées
 - Les algorithmes sont souvent présentés de manière procédurale.
 - Le hardware a un fonctionnement impératif.
-
- Implique un changement d'habitude (apprentissage)

Immutabilité

Utiliser des `val` à la place des `var` !

```
class PointM( var x: Double, var y: Double ) {  
  def moveHorizontaly( dx: Double ): Unit = {  
    x = x + dx  
  }  
}
```

```
case class PointI( x: Double, y: Double ) {  
  def moveHorizontaly( dx: Double ): PointI =  
    copy( x = x+dx )  
}
```

Boucles

Pas moyen d'avoir une boucle sans variable ou sans effet de bord !

```
def sum( is: Array[Int] ): Int = {  
  var i = 0  
  var sum = 0  
  while( i < is.size ) {  
    sum += is(i)  
    i += 1  
  }  
  sum  
}
```

Récursion

```
def sum( is: Array[Int], i: Int = 0 ): Int =  
  if( i == is.size ) 0  
  else {  
    is(i) + sum(is,i+1)  
  }  
  
val s = sum( is )
```

Récursion terminale

```
def sum( is: Array[Int] ): Int = {  
  
    def sumRec( i: Int, sum: Int ): Int =  
        if( i == is.size ) sum  
        else sumRec( i+1, sum+is(i) )  
  
    sumRec( 0, 0 )  
}
```

Récursion terminale

- Le résultat est accessible à la fin de la récursion.
- Compilé sous la forme d'une boucle `while`:
 - rapide
 - pas de `StackOverflowError`
- La méthode ne doit pas être héritée: méthode imbriquée, `final` ou `private`.
- L'annotation `@tailrec` permet de vérifier que la récursion est bien terminale.

`@annotation.tailrec`

```
def sumRec( i: Int, sum: Int ): Int =  
  if( i == is.size ) sum  
  else sumRec( i+1, sum+is(i) )
```

Pile Procédurale: Mettre à jour l'état

```
trait StackM[A] {  
  
  def isEmpty: Boolean  
  
  def push( a: A ): Unit  
  
  def pop: A  
  
}
```

Tuples

Collection **immutable** de données de types différents:

```
val x = (10,"ten")           //(Int,String)
val y = (true,true,false,2) //(Boolean,Boolean,Boolean,Int)

println( x._1 + " " + x._2 )

val (num,english) = x
val (_,b,_i) = y
```


Tuples (remarques)

- Au maximum 22 éléments
- Implémentés par les classes TupleN
- 1->"one"= est synonyme de =(1,"one")
- Commode dans un match case:

```
(x,y) match {  
  case (-1,2) => "foo"  
  case (_,0) => "bar"  
  case _ => "baz"  
}
```

Pile Fonctionnelle: Retourner le nouvel état

```
trait StackI[A] {  
  
  def isEmpty: Boolean  
  
  def push( a: A ): StackI[A]  
  
  def pop: (A, StackI[A])  
  
}
```

Exemples

```
def addTop( stack: StackM[Int] ): Unit = {  
  val x = stack.pop  
  val y = stack.pop  
  stack.push( x + y )  
}
```

```
def addTop( stack: StackI[Int] ): StackI[Int] = {  
  val (x,stack1) = stack.pop  
  val (y,stack2) = stack1.pop  
  stack2.push( x + y )  
}
```

ls