# Recursion

Jean-Luc Falcone

10 mars 2022

# Recursion Terminale

# Boucles

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

```
// Java, C, etc.
int dumbCount( int n ) {
  int c = 0;
  int m = n;
  while( n > 0 ) {
    c += 1;
    m -= 1;
  }
  return c;
}
```

# Récursion

```
def dumbCount1( n: Int ): Int =
  if( n == 0 ) 0 else 1 + dumbCount1(n-1)
```

# Récursion terminale: Définition

```
def dumby( m: Int, c: Int ): Int =
  if( m == 0) c else dumby(m-1,c+1)
```

Toutes les branches du code sont:

- Soit une valeur de retour (ici `c`)
- Soit un appel recursif seul (`dumby(m-1,c+1)`)

On ajoute des arguments pour stocker le résultat courant.

# Récursion terminale: remarques

- Optimisation: Compilé sous la forme d'une boucle while :
  - plus rapide
  - pas de `StackOverflowError`

- La méthode ne doit pas être redéfinie (`override`):
  - méthode imbriquée
  - `final`
  - `private`.

# Annotation

- Le compilateur applique l'optimisation automatiquement dès que possible.
- L'annotation `tailrec` permet de vérifier que l'optimisation est bien appliquée.

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

# Méthode imbriquée

Permet non seulement de garantir que la méthode ne peut pas être redéfinie mais permet de cacher les paramètres supplémentaires.

```
def dumbCount1( n: Int ): Int =
  if( n == 0 ) 0 else 1 + dumbCount1(n-1)

def dumbCount2( n: Int ): Int = {
  def dumby( m: Int, c: Int ): Int =
    if( m == 0) c else dumby(m-1,c+1)
  dumby( n, 0 )
}
```

# Recursion terminale

```
def sqrSum( xs: Array[Double] ): Double = {
  var s = 0.0
  var i = 0
  while( i < xs.length ) {
    s += xs(i) * xs(i)
    i += 1
  }
  s
}
```

```
def sqrSum2( xs: Array[Double] ): Double = {

  def loop( i: Int, s: Double ): Double =
    if( i < xs.length )
      loop( i + 1, s + xs(i)*xs(i) )
    else s

  loop( 0, 0.0 )
}
```