

Programmation Fonctionnelle

Premiers pas en terrain récuratif et itératif

Gilles Enée – LS4

Université des Antilles Guyane



Programmer par récurrence

- La récurrence est un outil d'une puissance inégalée pour :
 - Construire des objets [Un entier n'est pas autre chose que 0 ou le successeur d'un entier naturel].
 - Démontrer des théorèmes : $1^2+2^2+\dots+n^2 = n(n+1)(2n+1) / 6$
 - Programmer ! C'est bien entendu ce qui va nous intéresser ici.

Programmer par récurrence

- Encore que programmer, c'est construire ... une **fonction récursive**.
- Ce mot savant désigne simplement une fonction définie par récurrence, i.e. faisant appel à elle-même.
- Exemple pour la factorielle d'un entier naturel :
 - $n! = 1$ si $n = 0$
 - $n! = n(n-1)!$ sinon

Programmer par récurrence

- Ce que nous traduirons naturellement en SCHEME par :

(define (fac n) ; n entier naturel, calcule n!

(if (zero? n)

1

; le cas de base

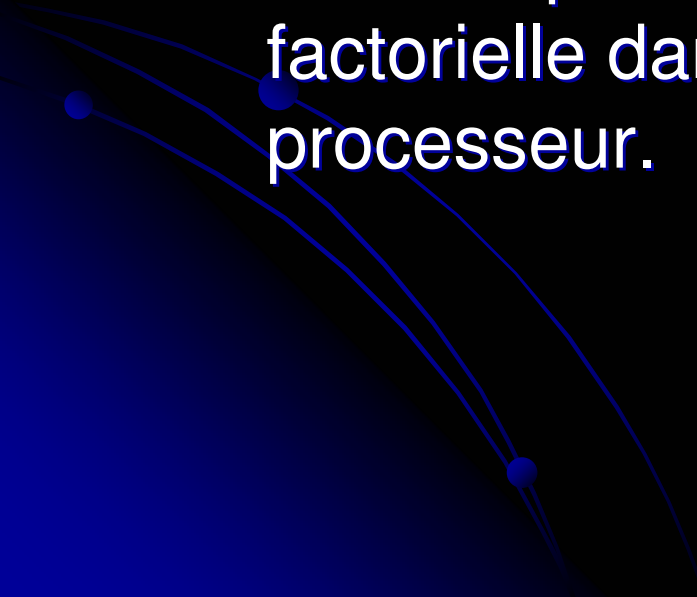
(* n (fac (- n 1))))

; le cas général

Programmer par récurrence

- Vous demanderez au toplevel 500!
- On peut chronométrer un algorithme grâce à *time* (à faire en TP):
(time (fac 500))
- Vous venez de voir une description axiomatique de la factorielle.
- Vous ne devez pas chercher à savoir comment le calcul se déroule dans la machine !

Programmer par récurrence

- Pourquoi ?
 - Parce que le temps d'exécution va dépendre de l'état de l'art des compilateurs.
 - On peut imaginer qu'un très bon compilateur divisera par 10 le temps d'exécution de cette factorielle dans 10 ans avec le même processeur.
- 

Programmer par récurrence

- Il vous faut raisonner de manière statique : supposons que le programme marche bien pour $n-1$ et démontrons [i.e. faisons en sorte] qu'il marche bien pour n .
- Ne cherchez pas à penser dynamiquement : comment va-t-il calculer, va-t-il faire une boucle, etc ?
- On s'en ~~rien~~ moque, c'est son problème.

Programmer par récurrence

- Notre problème, c'est la rigueur de la récurrence.
- En d'autres termes, oubliez les boucles, les stratégies et les actions, et raisonnez comme en maths, n'hésitez pas à poser une hypothèse de récurrence si vous êtes bloqué.

Programmer par récurrence

- Retenez :
 - « Une fonction récursive se construit comme une démonstration par récurrence :
 - On commence par traiter le cas général en trouvant une relation de récurrence $n! = n(n-1)!$
 - On s'aperçoit qu'il y a une quantité strictement décroissante qui va converger, ici n vers 0 : le cas de base.

N.B. L'oubli du cas de base a des conséquences terribles, vous devinez pourquoi ?

Programmer par récurrence

- Au lieu de faire une récurrence de $n-1$ vers n . Ray Cursif propose d'aller de n vers $n+1$ en utilisant la relation :

$$n! = (n + 1)! / (n+1)$$

```
(define (ray-fac n)
```

```
  (if (zero? n)
```

```
      1
```

```
      (quotient (ray-fac (+ n 1)) (+ n 1))))
```

- Faites une trace avec : `(ray-fac 5)`
 - Qu'est-ce qui se passe ?
 - Vous vous essaieriez aussi en TP avec les boutons de trace.

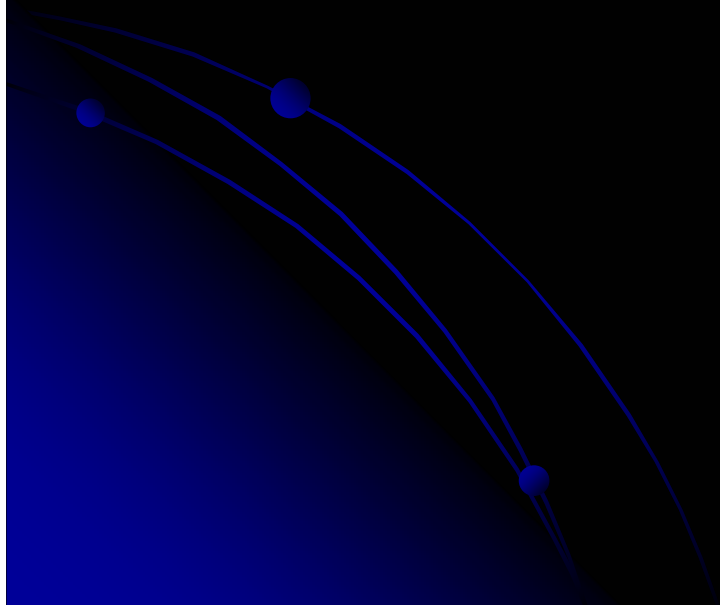
Programmer itérativement

- Kézako ?
 - Il s'agit de déplacer la récurrence et d'accumuler le calcul.
 - On se sert d'une fonction auxiliaire définie dans le corps qui va accumuler le calcul.
 - On accumule le calcul au lieu de retourner un calcul.
 - On se sert d'autant de variable de calcul que nécessaire (en général que de terme de la récurrence)

Programmer itérativement

```
(define (fac n)
  (if (zero? n)
      1
      (* n (fac (- n 1)))))
```

```
(define (fac n)
  (define (iter acc n)
    (if (zero? n)
        acc
        (iter (* acc n) (- n 1))))
  (iter 1 n))
```



Programmer itérativement

- Programmer par récurrence et itérativement la suite de Fibonacci :

$$\text{fib}(n) = n \text{ si } n < 2$$

$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2) \text{ sinon}$$

- Quel est l'intérêt de l'une par rapport à l'autre ?

Créer sa propre bibliothèque

- Lorsque vous voulez avoir à disposition des fonctions que vous avez défini. Il suffit de les regrouper dans un fichier Scheme qui servira de bibliothèque chargeable à volonté. Il suffit alors d'appeler ce fichier à l'aide de la commande load en début de définition :
(load "Monfichier.scm")