# TD de programmation fonctionnelle et logique

# Corrigé du TD 3 : récursivité

## Suite de Fibonacci

La suite de Fibonacci est définie comme suit :

$$\text{Fib}(n) = \begin{cases} 1 \text{ si } n = 0 \\ 1 \text{ si } n = 1 \\ \text{Fib}(n-1) + \text{Fib}(n-2) \text{ sinon.} \end{cases}$$

1. Écrire une fonction récursive qui calcule $\text{Fib}(n)$.

```
# let rec fib_filtre =
   function
      0 -> 1
    | 1 -> 1
    | n -> (fib_filtre (n-1)) + (fib_filtre (n-2));;
val fib_filtre : int -> int = <fun>
```

2. Écrire une fonction récursive qui calcule, pour $n > 0$, $(\text{Fib}(n), \text{Fib}(n-1))$. L'utiliser pour écrire une fonction calculant $\text{Fib}(n)$.

```
# let rec fib_paire =
   function
      0 -> failwith "L'argument est nul."
    | 1 -> (1,1)
    | n -> let (x,y)= fib_paire (n-1) in(x+y,x);;
val fib_paire : int -> int * int = <fun>
```

```
# let fib = function
      0 -> 1
    | n -> let (x,y)= fib_paire n in x;;
val fib : int -> int = <fun>
```

## Calcul de (cos(nx), sin(nx))

Écrire une fonction qui prend en entrée un entier `n` et une paire de valeurs réelles qui sont en fait les valeurs du cosinus et du sinus d'un certain angle `x`, et qui renvoie la paire `(cos(nx), sin(nx))`. Autrement dit, le deuxième argument de la fonction est une paire `(a,b)` telle que `a = cos x` et `b = sin x` . Le schéma de calcul doit bien évidemment être récursif.

On pourra se servir des formules de trigonométrie suivantes :

```
cos(nx) = cos((n-1)x) cos(x) - sin((n-1)x) sin(x)
sin(nx) = sin((n-1)x) cos(x) + cos((n-1)x) sin(x)
```

```
# let trigo (a, b) n =
   let rec trigo_aux = function
       0 -> (1.,0.)
     | 1 -> (a, b)
```

```
    | m -> let (c,d) = trigo_aux (m-1)
             in (a *. c -. b *. d, a *. d +. b *. c)
  in trigo_aux n;;
val trigo : float * float -> int -> float * float = <fun>
```

## Tri par recherche du maximum

1. Écrire une fonction qui retourne la valeur du maximum d'une liste.

```
# let rec maximum = function
    []   -> failwith "Liste vide"
  | [a]  -> a
  | t::r -> max t (maximum r);;
val maximum : 'a list -> 'a = <fun>
```

2. Écrire une fonction qui prend une liste en entrée et retourne la liste privée de son élément maximal.

   (a) Écrire une version de cette fonction qui recherche d'abord l'élément maximum, puis qui l'enlève de la liste.

```
#    maximum [3;2;4];;
- : int = 4

# let rec ote x = function
   t::r -> if (t=x) then r
                    else t::(ote x r)
  |  []  -> [];;
val ote : 'a -> 'a list -> 'a list = <fun>

# let sansmax l = ote (maximum l) l;;
val sansmax : 'a list -> 'a list = <fun>

# let sansmax l =
   let rec ote x = function
     t::r -> if (t=x) then r
                      else t::(ote x r)
    |  []  -> []
   in ote (maximum l) l;;
val sansmax : 'a list -> 'a list = <fun>

# sansmax [2;7;4;6];;
- : int list = [2; 4; 6]
```

   (b) Écrire une version de cette fonction qui calcule simultanément le maximum et la liste demandée.

```
# let sansmax =
   let rec aux x = function
        []  -> []
     | t::r -> if (t>x) then (x::(aux t r))
                        else t::(aux x r)
   in function    []  -> []
               | t::r -> aux t r;;
val sansmax : 'a list -> 'a list = <fun>

# sansmax [2;7;4;6];;
- : int list = [2; 4; 6]
```

3. Si les paires ont été vues en cours, écrire une fonction qui prend en entrée une liste et renvoie une paire composée de l'élément maximum et de la liste des autres éléments.

```
# let maxpaire =
   let rec aux x accu = function
```

```
        []  -> (x,accu)
     | t::r -> if (t>x) then aux t (x::accu) r
                       else aux x (t::accu) r
   in function   []  -> failwith "Liste vide"
              | t::r -> aux t [] r;;
val maxpaire : 'a list -> 'a * 'a list = <fun>

#maxpaire [2;7;6;4];;
- : int * int list = 7, [4; 6; 2]
```

4. Écrire une fonction de tri de liste par recherches successives de maximum (on utilisera bien évidemment certaines des fonctions précédemment obtenues).

   (a) Écrire une fonction qui trie dans l'ordre décroissant.

```
#let rec tri = function
     [] -> []
   |  l -> let x = maximum l and liste = sansmax l
                in x::(tri liste);;
val tri : 'a list -> 'a list = <fun>

#tri [2;7;6;4];;
- : int list = [7; 6; 4; 2]

#let rec tri = function
     [] -> []
   |  l -> let (x,liste) = maxpaire l
                in x::(tri liste);;
val tri : 'a list -> 'a list = <fun>

#tri [2;7;6;4];;
- : int list = [7; 6; 4; 2]
```

   (b) Écrire une fonction qui trie dans l'ordre croissant (sans utiliser de fonctions de concaténations de listes).

```
#let tri l =
    let rec aux accu = function
       []  -> accu
     |  _ as l -> let x = maximum l and liste = sansmax l
                  in aux (x::accu) liste
    in aux [] l;;
val tri : 'a list -> 'a list = <fun>

#tri  [2;7;6;4];;
- : int list = [2; 4; 6; 7]

#let tri l1 =
    let rec aux accu l2 = match l2 with
       [] -> accu
     |  _ -> let (x,liste) = maxpaire l2
             in aux (x::accu) liste
    in aux [] l1;;
val tri : 'a list -> 'a list = <fun>

#tri [2;7;6;4];;
- : int list = [2; 4; 6; 7]

#let tri =
    let rec aux accu = function
         []   -> accu
     | _ as l -> let (x,liste) = maxpaire l
                 in aux (x::accu) liste
```

```
   in aux [];;
val tri : '_a list -> '_a list = <fun>

# tri [2;7;6;4];;
- : int list = [2; 4; 6; 7]
```