Fonctionnalité et Modularité

Cours 2

Jean-Jacques Lévy

jean-jacques.levy@inria.fr

http://jeanjacqueslevy.net/prog-fm

Plan

- fonctions anonymes
- types polymorphes
- Ocaml: librairie standard
- exceptions
- alias
- n-uplets
- récursivité

dès maintenant: télécharger Ocaml en `http://www.ocaml.org`

Rappels et exercices

Exercice 1 Compter le nombre de zéros dans un tableau d'entiers

```
let add1_zero r x = if x = 0 then r + 1 else r ;;

let count_zeros = Array.fold_left add1_zero 0 ;;
```

Exercice 2 Mutiplier par 10 tous les éléments d'un tableau d'entiers

```
let mult10 x = x * 10;;

let mult10_array = Array.map mult10 ;;

let mult10_matrix = Array.map mult10_array ;;
```

Exercice 3 Créer l'image miroir d'un tableau d'entiers

```
let miroir_elt a n i x = a.(n - 1 - i) ;;

let miroir a =
  let n = Array.length a in
  Array.mapi (miroir_elt a n i) a ;;
```

les sources sont en http://jeanjacqueslevy.net/prog-fp/progs/fp3a.ml

Fonctions anonymes

Exercice 2 Mutiplier par 10 tous les éléments d'un tableau d'entiers

```
let mult10_array = Array.map (fun x -> x * 10);;

let mult10_matrix = Array.map (Array.map (fun x -> x * 10)) ;;
```

Exercice 1 Compter le nombre de zéros dans un tableau d'entiers

```
let count_zeros = Array.fold_left (fun r x -> if x = 0 then r + 1 else r) 0 ;;
```

Exercice 3 Créer l'image miroir d'un tableau d'entiers

```
let miroir a =
    let n = Array.length a in
    Array.mapi (fun i _ ->  a.(n-1 - i)) a ;;
```


```
let miroir1 a =
    let n = Array.length a in
    Array.init n (fun i -> a.(n-1 - i)) ;;
```

les sources sont en http://jeanjacqueslevy.net/prog-fp/progs/fp3a.ml

Fonctions anonymes

- déclaration d'une fonction anonyme avec le mot clé `fun`

```
fun x -> expression
```

- aussi avec plusieurs arguments

```
fun x1 x2 .. xn -> expression
```

- exemples

```
fun x -> x + 1

fun x y -> x + y

fun x y -> if x = y then 1 else 0

fun x y -> x ^ y
```

- exemples

```
(* val add : int -> int -> int = <fun> *)

let add = fun x y -> x + y ;;
```


```
let add x y = x + y ;;
```

Fonctions anonymes

Exercice 3 Créer l'image miroir d'un tableau d'entiers

```
let miroir a =
    let n = Array.length a in
    Array.mapi (fun i _ ->  a.(n-1 - i)) a ;;
```

```
let miroir1 a =
    let n = Array.length a in
    Array.init n (fun i -> a.(n-1 - i)) ;;
```

- quel est le type de `miroir` ?

```
let a = [| 3; 2; 7; 8; 1; 12; 30; 4; 2; 12 |] ;;
miroir a ;;
- : int array = [|12; 2; 4; 30; 12; 1; 8; 7; 2; 3|]

let b = [| "Bonjour"; " "; "tout le monde" |] ;;
miroir b ;;
- : string array = [|"tout le monde"; " "; "Bonjour"|]

let c = [| 'a'; 'b'; 'c';'d' |] ;;
miroir c ;;
- : char array = [|'d'; 'c'; 'b'; 'a'|]

let m = [| [| 1; 2 |] ; [| 3 ; 4 |] |] ;;
miroir m ;;
- : int array array = [|[|3; 4|]; [|1; 2|]|]
```

- le type de `miroir` est α array $\rightarrow \alpha$ array où α est un type quelconque

```
miroir ;;
- : 'a array -> 'a array = <fun>
```

`miroir`

fonction polymorphe

Types polymorphes

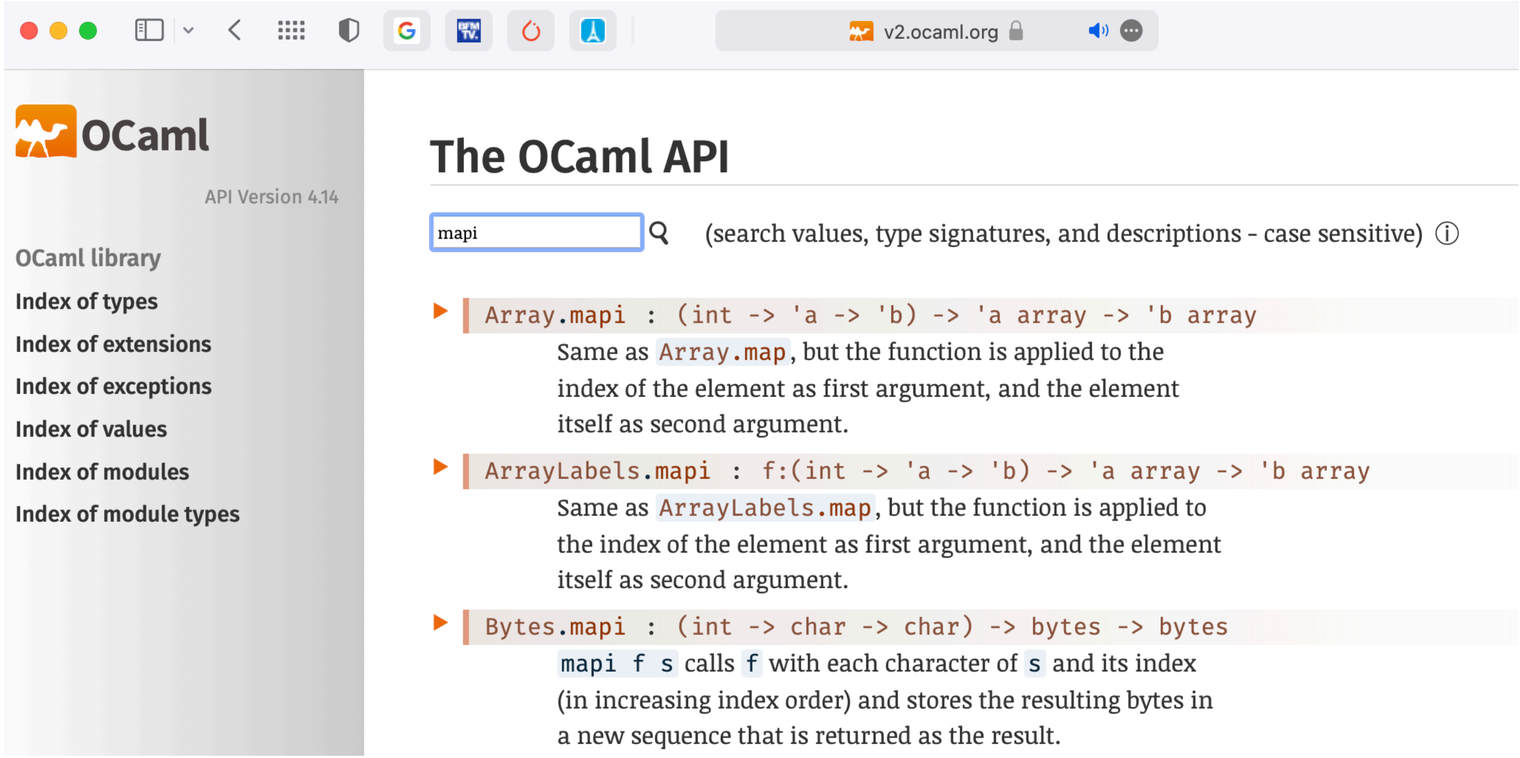
- exemples de fonctions polymorphes

```
Array.length ;;
- : 'a array -> int = <fun>

Array.init ;;
- : int -> (int -> 'a) -> 'a array = <fun>

Array.map ;;
- : ('a -> 'b) -> 'a array -> 'b array = <fun>

Array.mapi ;;
- : (int -> 'a -> 'b) -> 'a array -> 'b array = <fun>

Array.iter ;;
- : ('a -> unit) -> 'a array -> unit = <fun>

Array.iteri ;;
- : (int -> 'a -> unit) -> 'a array -> unit = <fun>

Array.fold_left ;;
- : ('a -> 'b -> 'a) -> 'a -> 'b array -> 'a = <fun>
```

- les types polymorphes évitent de réécrire la même fonction pour des types différents

Librairie standard

- l'API de Ocaml est visible en `http://v2.ocaml.org/api`
- avec les fonctions de la librairie standard aussi en `http://v2.ocaml.org/manual/stdlib.html`



Autres fonctions polymorphes

- dans le module `Array` de la librairie standard

```
val iter2 : ('a -> 'b -> unit) -> 'a array -> 'b array -> unit
val map2 : ('a -> 'b -> 'c) -> 'a array -> 'b array -> 'c array
val for_all : ('a -> bool) -> 'a array -> bool
val exists : ('a -> bool) -> 'a array -> bool
```

- somme et max de 2 tableaux

```
let a = [|3; 2; 7; 8; 1; 12; 30; 4; 2; 12|] ;;
let b = [|-3; 20; 4; 88; -1; 112; 300; -4; -2; 16|] ;;

let add2 = Array.map2 (+) ;;
let max2 = Array.map2 max ;;
```

- test positif / négatif dans un tableau

```
let is_negative_in a = Array.exists (fun x -> x < 0) a ;;
let is_negative_in = Array.exists ((>) 0) ;;
let all_positive_in = Array.for_all ((<=) 0) ;;
```

les sources sont en http://jeanjacqueslevy.net/prog-fp/progs/fp3a.ml

Exceptions

- utiliser des exceptions pour rompre une évaluation
- exemple: trouver l'indice d'un élément de valeur **v** dans le tableau *a* (ou -1 si **v** n'est pas dans *a*)

```
let index_in v a =
  let exception Found of int in
  try
    Array.iteri (fun i x -> if x = v then raise (Found i)) a;
    -1
  with Found i -> i ;;
```

on lève l'exception si on a trouvé **v**

- on déclare une exception `Found` qui a un paramètre entier (`int`)
- et on lève l'exception que l'on peut récupérer avec `try` .. `with`

les sources sont en `http://jeanjacqueslevy.net/prog-fp/progs/fp3a.ml`

Exceptions

Exercice 1 Trouver l'indice du maximum dans un tableau d'entiers

Exercice 2 Trouver l'indice du premier nombre négatif dans un tableau d'entiers

Exercice 3 Trouver l'indice du dernier nombre négatif dans un tableau d'entiers

Exercice 4 Trouver l'indice du premier caractère différent dans 2 chaînes `s` et `s'` (-1 si les mêmes chaînes)

[indication: utiliser la fonction `String.iteri`]

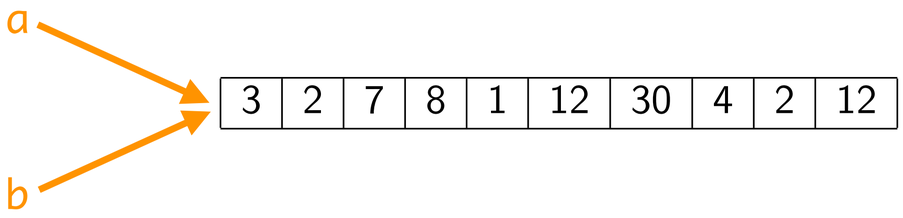
Valeur d'un tableau — Alias

- la fonction (polymorphe) suivante change la valeur d'un élément d'un tableau à un certain indice

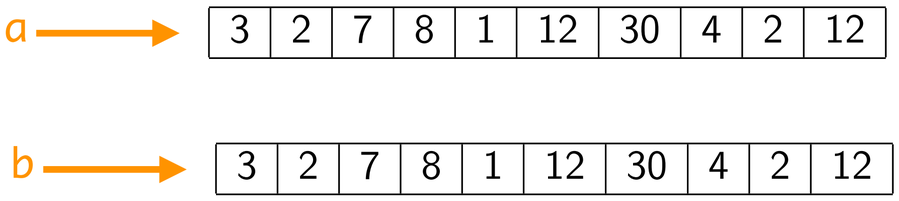
```
(* val store : int -> 'a -> 'a array -> unit = <fun> *)

let store i v a =  a.(i) <- v ;;
```

- soient 2 tableaux *a* et *b*

```
let a = [|3; 2; 7; 8; 1; 12; 30; 4; 2; 12|] ;;
let b = a ;;
store 2 888 a ;;
a ;;
- : int array = [|3; 2; 888; 8; 1; 12; 30; 4; 2; 12|]
b ;;
- : int array = [|3; 2; 888; 8; 1; 12; 30; 4; 2; 12|]
```



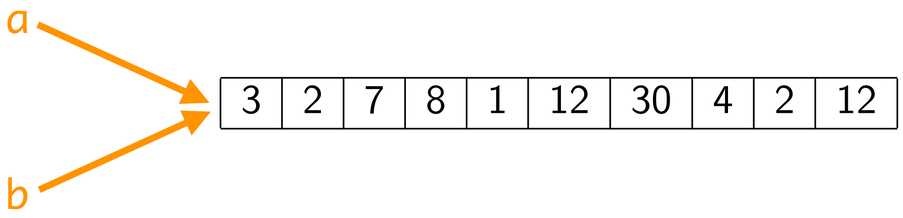

```
let a = [|3; 2; 7; 8; 1; 12; 30; 4; 2; 12|] ;;
let b = [|3; 2; 7; 8; 1; 12; 30; 4; 2; 12|] ;;
store 2 888 a ;;
a ;;
- : int array = [|3; 2; 888; 8; 1; 12; 30; 4; 2; 12|]
b ;;
- : int array = [|3; 2; 7; 8; 1; 12; 30; 4; 2; 12|]
```

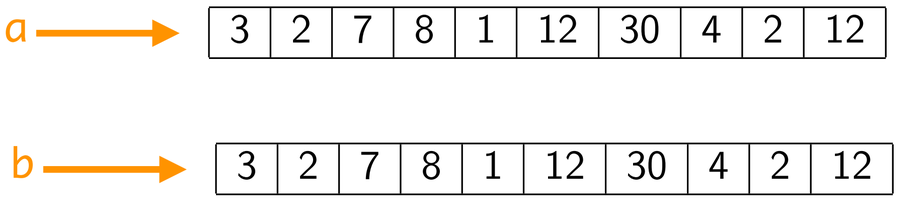


Valeur d'un tableau — Alias

- les 2 tableaux *a* et *b* sont des alias
- les valeurs de *a* et *b* sont les adresses (mémoire) où se trouvent ces tableaux
- **modification de la mémoire et alias ne font pas bon ménage**
- ce sont des sources de bugs
- la programmation fonctionnelle essaie de les éviter puisque toutes les variables sont des constantes

en Python, toutes les sont modifiables !!





n-uplets et chaînes

- les n-uplets (*tuples*)

```
(* val fete_nationale : int * string = (14, "juillet") *)

let fete_nationale = (14, "juillet") ;;

fst fete_nationale ;;
- : int = 14

snd  fete_nationale ;;
- : string = "juillet"

(* val bastille : int * string * int = (14, "juillet", 1789) *)

let bastille = (14, "juillet", 1789) ;;
```

- itérateurs sur les chaînes de caractères

```
String.length, String.map, String.mapi, String.iter, String.iteri
```

- n-uplets et chaînes ne sont pas modifiables

Exercices

Exercice 1 Trouver l'indice du maximum dans un tableau d'entiers

```
let min_in_array a = Array.fold_left min max_int a ;;

let index_min_of a = index_in (min_in_array a) a ;;
```

Exercice 2 Trouver l'indice du premier nombre négatif dans un tableau d'entiers

```
let index_fst_neg_in a =
  let exception Found of int in
  try
    Array.iteri (fun i x -> if x < 0 then raise (Found i)) a;
    -1
  with Found i -> i ;;
```

Exercice 3 Trouver l'indice du dernier nombre négatif dans un tableau d'entiers

```
let index_lst_neg_in a =
  let n = Array.length a in
  n-1 - index_fst_neg_in (miroir a) ;;
```

les sources sont en http://jeanjacqueslevy.net/prog-fp/progs/fp4a.ml

Exercices

Exercice 4 Trouver l'indice du premier caractère différent dans 2 chaînes `s` et `s'` (-1 si les mêmes chaînes)

```
let index_diff a b =
  if String.length a = String.length b then
    let exception Found of int in
    try
      String.iteri (fun i _ -> if a.[i] != b.[i] then raise (Found i)) a;
      -1
    with Found i -> i
  else -1 ;;
```

les sources sont en http://jeanjacqueslevy.net/prog-fp/progs/fp4a.ml

Exercices

Fonctions utiles:

- imprimer un tableau d'entiers

```
let print_array a =
  Array.iter (Printf.printf "%d ") a;
  print_newline() ;;
```

- générer un tableau de `n` entiers avec des valeurs aléatoires entre `0` et `p-1`

```
let rand_array n p = Array.init n (fun i -> Random.int p) ;;
```

les sources sont en `http://jeanjacqueslevy.net/prog-fp/progs/fp4a.ml`

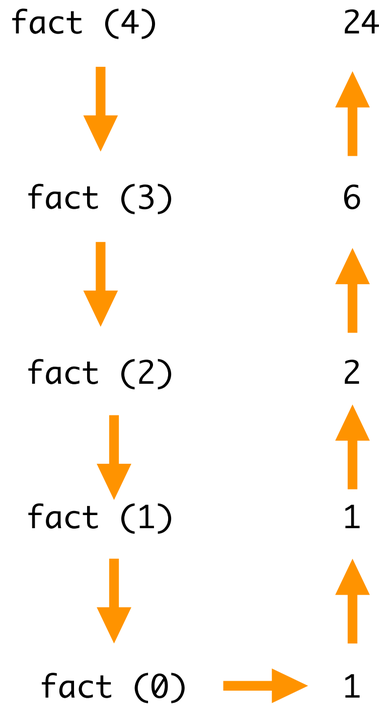
Fonctions récursives

- une fonction qui se rappelle avec un argument plus petit

```
let rec fact =
  if n = 0 then 1 else n * fact (n-1) ;;

fact 3 ;;

fact 10 ;;
```

factorielle

appels récursifs



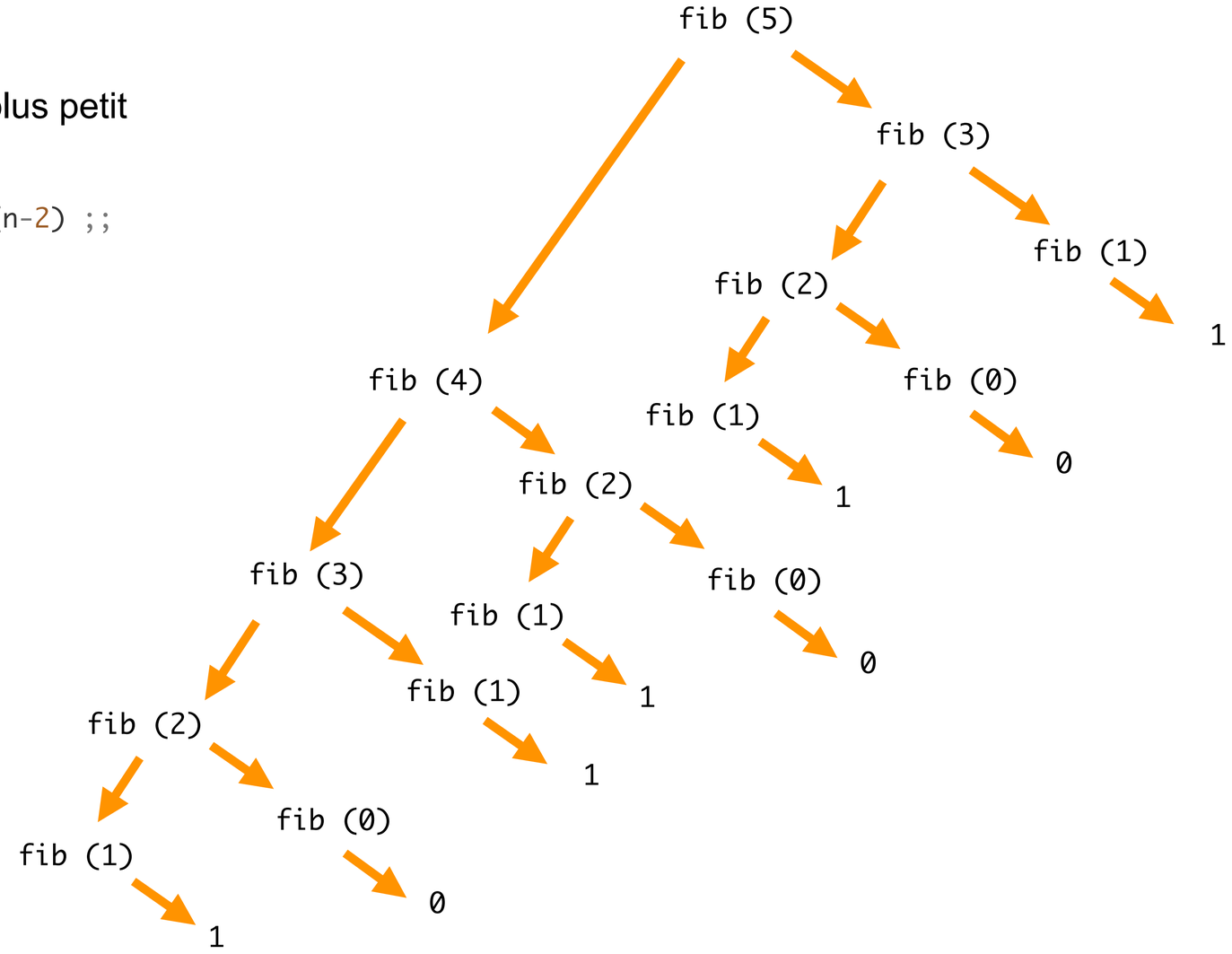
les sources sont en http://jeanjacqueslevy.net/prog-fp/progs/fp5a.ml

Fonctions récursives

- une fonction qui se rappelle avec un argument plus petit

```
let rec fib n =
  if n = 0 || n = 1 then n else fib (n-1) + fib (n-2) ;;

fib 10 ;;

fib 35 ;;
```

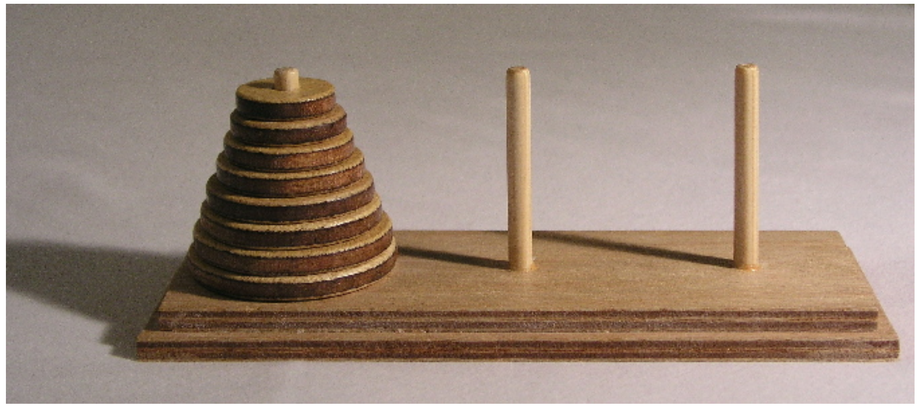
fibonacci



- écrire une fonction qui calcule `fibonacci` plus rapidement

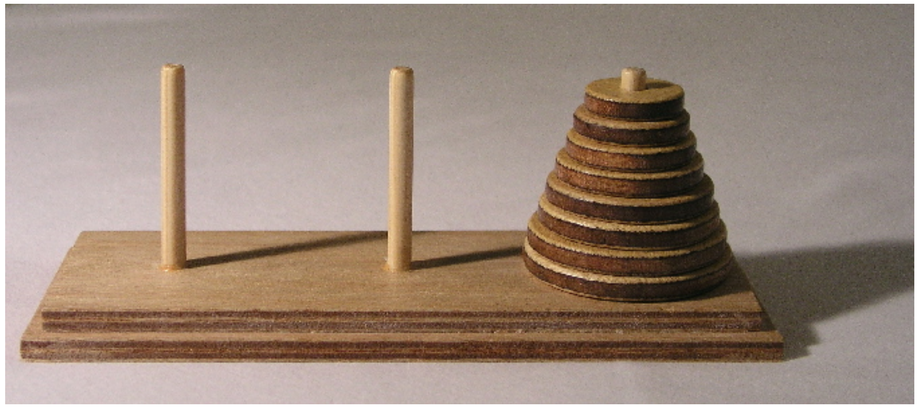
Les tours de Hanoi

- on a 3 piles et n rondelles sur la pile 1
- jamais une rondelle grosse au-dessus d'une rondelle petite



pile 1 pile 2 pile 3

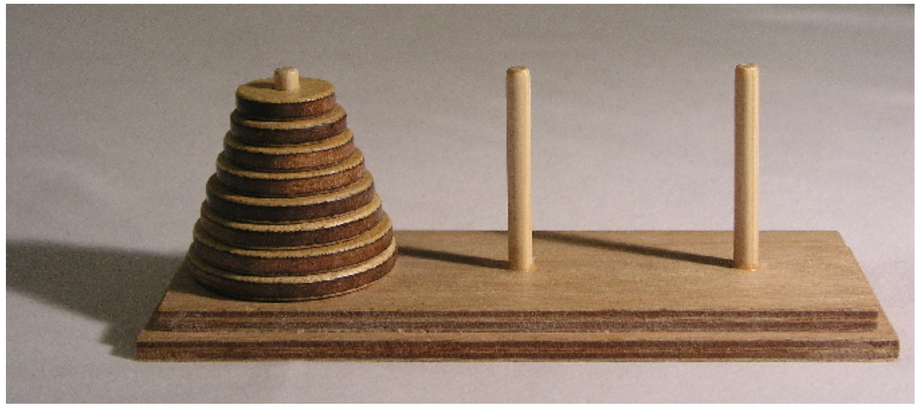
- il faut amener les n rondelles sur la pile 3
- on ne déplace qu'une seulle rondelle à la fois
- et on ne met jamais une rondelle au-dessus d'une plus petite



pile 1 pile 2 pile 3

Les tours de Hanoi

- on a 3 piles et n rondelles sur la pile 1
- jamais une rondelle grosse au-dessus d'une rondelle petite



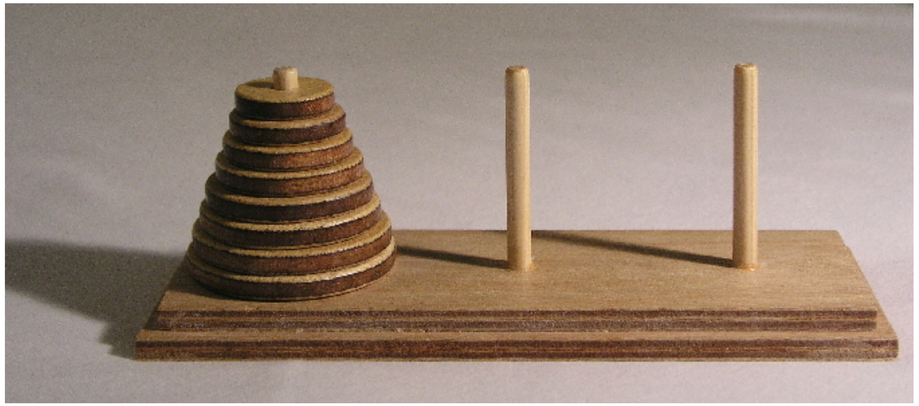
pile 1 **pile 2** **pile 3**

- il faut amener les n rondelles sur la pile 3
- on ne déplace qu'une seule rondelle à la fois
- et on ne met jamais une rondelle au-dessus d'une plus petite

Les tours de Hanoi

- on généralise le problème pour aller de la pile i à la pile j
  où $1 \leq i \leq 3$ et $1 \leq j \leq 3$

  la troisième pile est alors $6 - i - j$



pile i pile $6 - i - j$ pile j

- supposons le problème résolu pour n-1 rondelles entre pile i et pile j

- j'amène les n-1 rondelles du dessus de la pile i sur la troisième pile
- j'amène la grosse rondelle de la pile i vers la pile j
- j'amène les n-1 rondelles de la troisième pile vers la pile j

```
let rec hanoi n i j =
  if n > 0 then begin
    hanoi (n-1) i (6 - i -j) ;
    printf "%d --> %d\n" i j ;
    hanoi (n-1) (6 -i -j) j
  end ;;
```

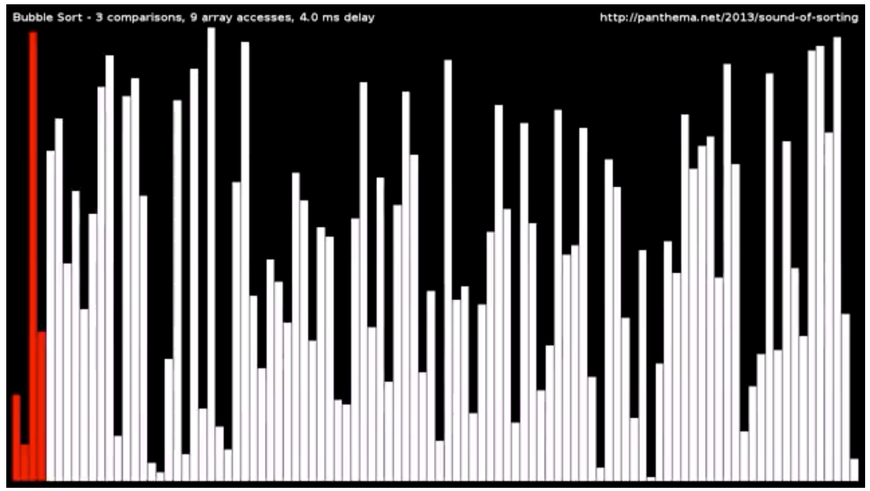
Tri bulle

- on veut trier les éléments d'un tableau d'entiers par ordre croissant

```
let bubble_sort a =
  let n = Array.length a in
  for i = n-1 downto 0 do
    for j = 0 to i-1 do
      if a.(j) > a.(j+1) then
        xchange a j (j+1)
    done
  done;;
```

```
let xchange a i j =
  let tmp = a.(i) in
  a.(i) <- a.(j);
  a.(j) <- tmp ;;
```

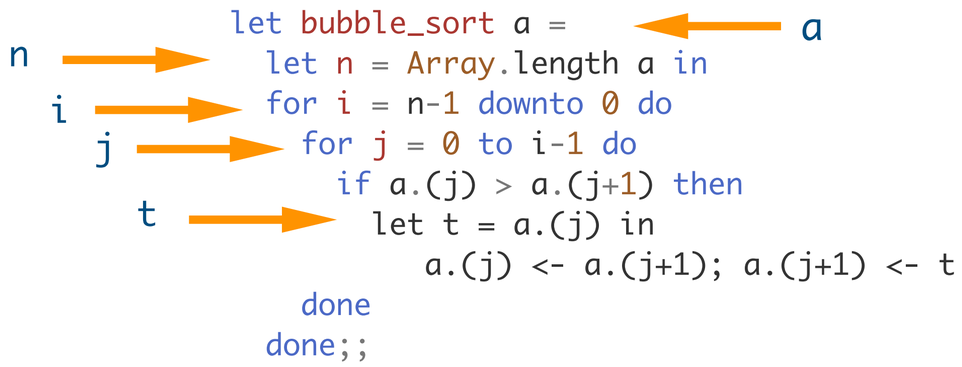
échanger les valeurs de `a.(i)` et `a.(j)`



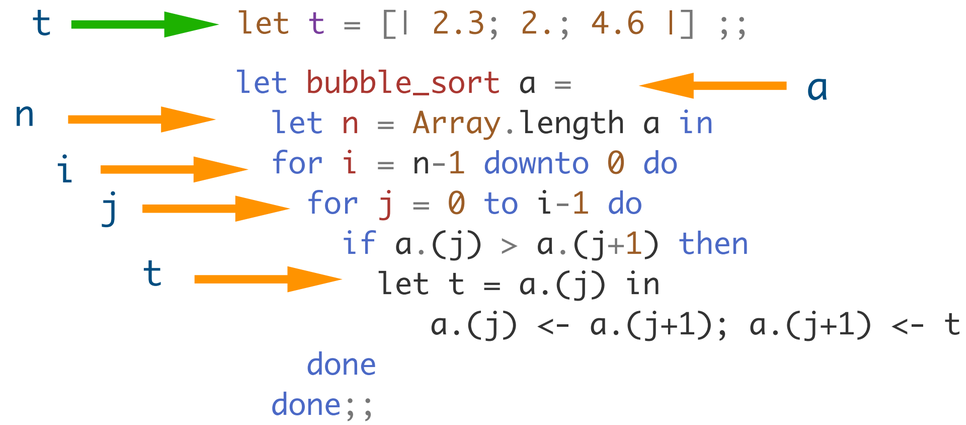
les sources sont en http://jeanjacqueslevy.net/prog-fp/progs/fp4a.ml

Quelques remarques

- les variables déclarées dans une fonction n'existent que dans le code de cette fonction



- les variables `n`, `j`, `i`, `t` et *a* sont **locales** à la fonction `tri_bulle`



- la variable `t` **globale** est distincte de la variable `t` locale

Quelques remarques

- les fonctions ou données (de librairie..) sont regroupées en modules

- notation qualifiée avec nom de module `Random.int`

```
let rand_array n p =
    Array.init n (fun i -> Random.int p) ;;
```

- on peut utiliser la notation simple `open_graph` sans le nom du module `Graphics` avec

```
open Graphics;;
```

- la liste des modules disponibles figure (en partie) dans la bibliothèque standard

`http://v2.ocaml.org/manual/stdlib.html`

Conclusion

VU:

- fonctions anonymes
- types polymorphes
- exceptions
- alias
- n-uplets et chaînes de caractères
- récursivité

TODO list

- références et variables modifiables
- types de données
- listes
- filtrage