Fonctionnalité et Modularité

Cours 1

Jean-Jacques Lévy

jean-jacques.levy@inria.fr

http://jeanjacqueslevy.net/prog-fm

Plan

- fonctions
- données scalaires
- tableaux
- itérations sur les tableaux
- palindrome, carré magique
- alias

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

Débuter en Ocaml

- utiliser un système intégré (Visual Studio ou autre)
- ou utiliser une simple fenêtre terminal [le plus simple]
- avec un éditeur de texte (Emacs, VI, TextEdit, ..)

exemple avec Ocaml

- sur la fenêtre terminal, on tape:

```
mac$ ocaml
        OCaml version 4.11.0

#
```

- et on peut fonctionner en mode calculette:

```
# 23 + 42 ;;
- : int = 65

# 438 * 234 ;;
- : int = 102492

# (438 * 234) + 35 ;;
- : int = 102527


# 5.0 ;;
- : float = 5.

# 5 ;;
- : int = 5
```

int (type entier)

float (type réel flottant)

Un programme droite-ligne

- date de Pâques

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

let paques y  =
      let g = (y mod 19) + 1 in
      let c = y / 100 + 1 in
      let x = 3 * c / 4 - 12 in
      let z = (8 * c + 5) / 25 - 5 in
      let d = 5 * y / 4 - x - 10 in
      let e = (11 * g + 20 + z - x) mod 30 in
      let e' =  if  e = 25 && g > 11 || e = 24 then
          e + 1
      else e in
      let n = 44 - e' in
      let n' = if n < 21 then
            n + 30
      else n in
      let j = n' + 7 - ((d + n') mod 7) in
      if j > 31 then
        Printf.printf "%d %s %d\n" (j - 31) "avril" y
      else
        Printf.printf "%d %s %d\n" j "mars" y
      ;;
```

e' est une nouvelle variable

n' est une nouvelle variable

- en Python

```
def paques (y) :
    g = (y % 19) + 1
    c = y // 100 + 1
    x = 3 * c // 4 - 12
    z = (8 * c + 5) // 25 - 5
    d = 5 * y // 4 - x - 10
    e = (11 * g + 20 + z - x) % 30
    if  e == 25 and g > 11 or e == 24 :
        e = e + 1
    n = 44 - e
    if n < 21 :
        n = n + 30
    j = n + 7 - ((d + n) % 7)
    if j > 31 :
        print ("%d avril %d" %(j - 31, y))
    else :
        print ("%d mars %d" %(j, y))
```

la valeur de e est modifiée

la valeur de n est modifiée

- dans les langagess fonctionnels, la valeur des variables est constante

Rappels

• définir des fonctions

```
(* val cat : string -> string -> string = <fun> *)

let cat s1 s2 = s1 ^ s2 ;;

cat "Bonjour " "les amis !" ;;
```

• la valeur d'une variable peut être une fonction

```
(* val hello : string -> string = <fun> *)
let hello = cat "Bonjour " ;

hello "Jean-Jacques" ;;
hello "les amis!" ;;
```

la valeur de hello est une fonction

pas aussi simple en Python !

• dans les langagess fonctionnels, les fonctions sont des valeurs comme les autres

• source en http://jeanjacqueslevy.net/

Tableaux

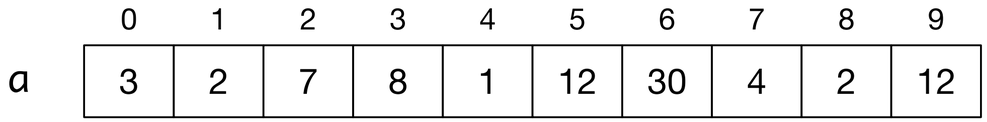
• données scalaires: entiers, flottants, booléens, caractères, chaînes de caractères

• données structurées: chaînes de caractères, listes, tableaux, etc..

```
(* val a : int array = [|3; 2; 7; 8; 1; 12; 30; 4; 2; 12|] *)

let a = [| 3; 2; 7; 8; 1; 12; 30; 4; 2; 12 |] ;;

a.(0) ;;
- : int = 3
a.(1) ;;
- : int = 2
a.(4) ;;
- : int = 1
```



• les valeurs des éléments d'un tableau sont modifiables

```
a.(7) <- 28 ;;
- : unit = ()

a ;;
- : int array = [|3; 2; 7; 8; 1; 12; 30; 28; 2; 12|]
```

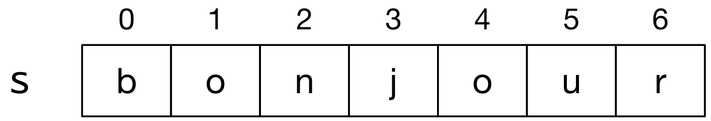
• les tableaux sont homogènes (éléments tous du même type)

Tableaux

- les chaînes de caractères sont des tableaux non modifiables de caractères

```
(* val s : string = "bonjour" *)
let s = "bonjour" ;;

s.[0] ;;
- : char = 'b'
s.[5] ;;
- : char = 'u'
```



Tableaux

• comment calculer le maximum d'un tableau puisque les valeurs des variables sont constantes ?

- méthode 1: utiliser la récursivité (voir plus tard)
- méthode 2: utiliser des références (voir plus tard)
- méthode 3: utiliser des itérateurs prédéfinis sur les tableaux

	0	1	2	3	4	5	6	7	8	9
a	3	2	7	8	1	12	30	4	2	12

• itérateurs sur les tableaux

```
a;;
- : int array = [|3; 2; 7; 8; 1; 12; 30; 28; 2; 12|]

let succ x = x + 1;;

Array.map succ a;;
- : int array = [|4; 3; 8; 9; 2; 13; 31; 29; 3; 13|]
```

la fonction succ est appliquée à tous les éléments du tableau a

peut s'exécuter en parallèle sur plusieurs processeurs

les sources sont en http://jeanjacqueslevy.net/prog-fp/progs/fp2b.ml

Tableaux

- itérateurs sur les tableaux

```
a;;
- : int array = [|3; 2; 7; 8; 1; 12; 30; 28; 2; 12|]

let add x y = x + y ;;

Array.fold_left add 0 a ;;
- : int = 105
```

	0	1	2	3	4	5	6	7	8	9
a	3	2	7	8	1	12	30	4	2	12

la fonction add est appliquée successivement
aux éléments du tableau a à partir de la valeur initiale 0

```
Array.fold_left max (-1) a;;
- : int = 30
```

la fonction max est appliquée successivement
aux éléments du tableau a à partir de la valeur initiale -1

souvent appelé « reduce »

- pas la peine de définir add

```
Array.fold_left (+) 0 a;;
- : int = 105
```

Tableaux

- la fonction sum_of s'écrit

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

let sum_of a = Array.fold_left (+) 0 a;;
```

- la fonction max_of s'écrit

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

let max_of a = Array.fold_left max (-1) a;;
```

- que fait exactement Array.fold_left ?

```
Array.fold_left f v0 [| a0; a1; a2; … an |]   ≡   (f … (f (f (f v0 a0) a1) a2) … an)
```

- que fait exactement Array.map ?

```
Array.map f [| a0; a1; a2; … an |]   ≡   [| f(a0); f(a1); f(a2); … f(an) |]
```

Tableaux

• les fonctions sum_of et max_of ont un argument fonctionnel (add, max)

• on pouvait aussi les écrire plus simplement

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

let sum_of = Array.fold_left (+) 0 ;

(* val max_of : int array -> int = <fun> *)

let max_of = Array.fold_left max (-1) ;;
```

les éléments de a sont supposés positifs

• dans les langages fonctionnels, les expressions peuvent retourner des fonctions

• si le tableau peut avoir des éléments négatifs, on écrit max_of

```
let max_of = Array.fold_left max min_int ;;
```

min_int est le plus petit nombre entier

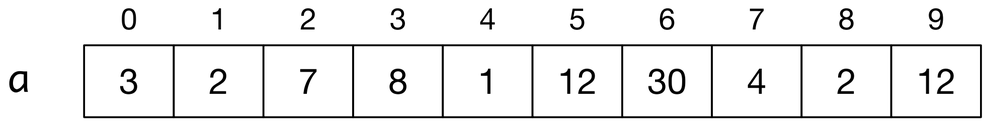
les sources sont en http://jeanjacqueslevy.net/prog-fp/progs/fp2b.ml

Tableaux

- il y a d'autres itérateurs

```
Array.fold_right (+) a 0;;
- : int = 105
```

`Array.iter`, `Array.iteri`, `Array.map`, `Array.mapi`, . . .



- le tout se trouve dans la librairie standard `http://v2.ocaml.org/manual/stdlib.html`

```
(* val print_array_int : int array -> unit = <fun> *)

let print_array_int  = Array.iter (Printf.printf "%d\n") ;;

print_array_int a ;;
2
7
8
1
12
30
4
2
12
- : unit = ()
```

```
Array.iter print_int a ;;
3278112304212- : unit = ()
```

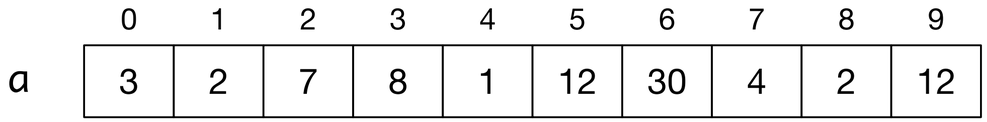
les sources sont en `http://jeanjacqueslevy.net/prog-fp/progs/fp2b.ml`

Tableaux

- il y a d'autres itérateurs

```
Array.fold_right (+) a 0;;
- : int = 105
```

`Array.iter`, `Array.iteri`, `Array.map`, `Array.mapi`, . . .



- le tout se trouve dans la librairie standard `http://v2.ocaml.org/manual/stdlib.html`

```
Array.iteri (Printf.printf "%2d: %d\n") a ;;
 0: 3
 1: 2
 2: 7
 3: 8
 4: 1
 5: 12
 6: 30
 7: 4
 8: 2
 9: 12
- : unit = ()
```

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

let zero_impair i x = if i mod 2 != 0 then 0 else x ;;

Array.mapi zero_impair a ;;
- : int array = [|3; 0; 7; 0; 1; 0; 30; 0; 2; 0|]
```

les sources sont en `http://jeanjacqueslevy.net/prog-fp/progs/fp2b.ml`

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

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`]

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

Conclusion

VU:

- tableaux et chaînes de caractères
- itérateurs sur tableaux et chaînes
- tableaux multidimensionnels

TODO list

- fonctions anonymes
- listes
- récursivité
- filtrage
- références et variables modifiables