Programmation fonctionnelle et Parallélisme

Cours 2

Jean-Jacques Lévy

jean-jacques.levy@inria.fr

http://jeanjacqueslevy.net/prog-fp

Plan

- rappels et solutions des exercices
- impressions formattées
- tableaux
- itérations sur les tableaux
- palindrome, carré magique
- alias

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

Rappels

- 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
      ;;
```

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

let verif_paques y1 y2 =
    for y = y1 to y2 do
      paques y
    done ;;

verif_paques 2021 2045 ;;
```

- surface d'un cercle: $4\pi r^2$

```
(* val surface : float -> float = <fun> *)

let surface r = 4. *. Float.pi *. r *. r ;;

surface 9.0 ;;
```

les sources sont en http://jeanjacqueslevy.net/prog-fp/progs/fp2a.ml

Rappels

- 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

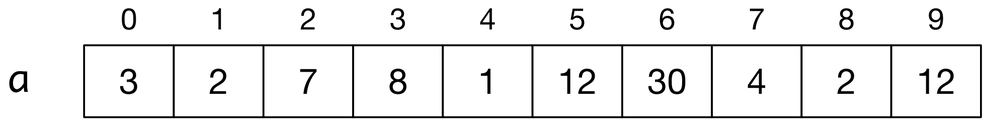
• données scalaires: entiers, flottants, booléens, caractères, chaînes de caractères

• 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|]
```

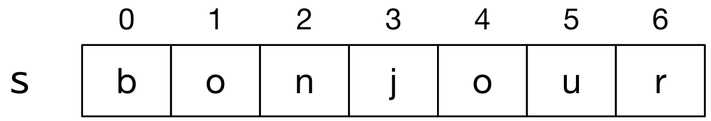
• les tableaux sont homogènes (éléments tous du même type)

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

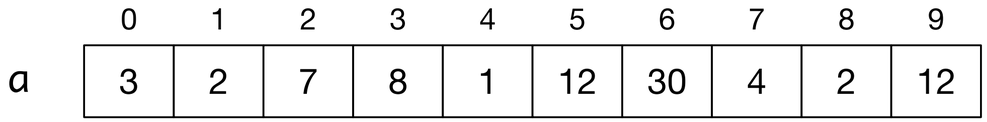
les sources sont en http://jeanjacqueslevy.net/prog-fp/progs/fp2b.ml

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 = ()
```

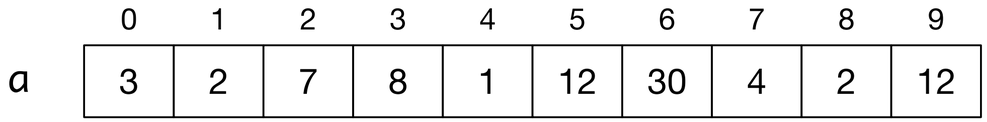
les sources sont en `http://jeanjacqueslevy.net/prog-fp/progs/fp2b.ml`

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|]
```

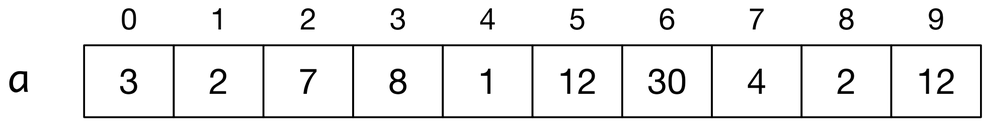
les sources sont en `http://jeanjacqueslevy.net/prog-fp/progs/fp2b.ml`

Tableaux

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

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

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



Tableaux multi-dimensionnels

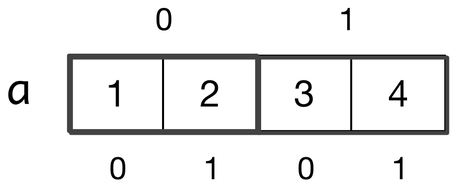
- une matrice est un tableau de tableaux

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

let a = [| [| 1; 2 |] ; [|3 ; 4 |] |];;

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

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$



- modification d'un élément

```
a.(0).(1) <- 22 ;;
- : unit = ()
```

- une itération sur les matrices

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

let print_row a = Array.iter (Printf.printf "%2d ") a ; print_newline() ;;

(* val print_matrix : int array array -> unit = <fun> *)
let print_matrix a = Array.iter print_row a ;;

print_matrix a;;
 1 22
 3  4
- : unit = ()
```

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

Tableaux multi-dimensionnels

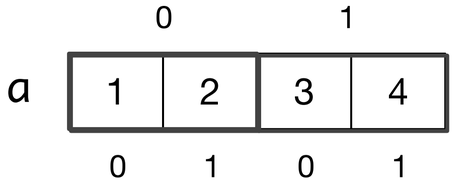
- une matrice est un tableau de tableaux

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

let a = [| [| 1; 2 |] ; [|3 ; 4 |] |];;

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

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$



- une autre itération sur les matrices

```
let double x = 2 * x;;

(* val double_row : int array -> int array = <fun> *)
let double_row = Array.map double;;

(* val double_matrix : int array array -> int array array = <fun> *)
let double_matrix = Array.map (Array.map double) ;;

double_matrix a;;
- : int array array = [|[|2; 44|]; [|6; 8|]|]

print_matrix (double_matrix a) ;;
 2 44
 6  8
```

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