petit mémo pour apprendre OCaml (resp. Java)
quand on connaît déjà Java (resp. OCaml)

Java	OCaml
variable locale	
`int x = 42;` (mutable par défaut)	`let x = 42 in` (immuable par défaut)

`int r = 41;`
`r = r + 1;`
	`let r = ref 41 in`
	`r := !r + 1`
	(variable mutable = référence)

tableaux	
`int[] a = new int[42];` (initialisé avec une valeur par défaut)	`let a = Array.make 42 0 in` (initialisé avec la valeur fournie, ici 0)

`a[17]`
`a[7] = 3;`
`a.length`
	`a.(17)`
	`a.(7) <- 3`
	`Array.length a`

enregistrements	

`class T {`
`  final int v; boolean b;`
`  T(int v, boolean b) {`
`  this.v = v; this.b = b;`
`  }`
`}`
(champ mutable, sauf si `final`)
	`type t = {`
	`  v: int;`
	`  mutable b: bool;`
	`}`
	(champ immuable, sauf si `mutable`)

`T r = new T(42, true);`
`r.b = false;`
`r.v`
	`let r = { v = 42; b = true }`
	`r.b <- false`
	`r.v`

fonctions	

`int fact(int x) {`
`  if (x <= 1) return 1;`
`  return x * fact(x-1);`
`}`
	`let rec fact x =`
	`  if x <= 1 then 1`
	`  else x * fact (x-1)`
	(le corps de la fonction est une expression)
	(en particulier, pas de `return`)

boucle for	
`for (int i = 0; i < n; i++) ...`	`for i = 0 to n-1 do ... done` (la variable `i` est immuable)
`for (X x: c) ...` (`c` collection d'éléments de type `X`)	`C.iter (fun x -> ...) c` (`c` venant d'un module `C` offrant `iter`)

Java	OCaml
arithmétique booléenne	
`a && b  a || b  !a`	`a && b  a || b  not a`
arithmétique	

`x % n`
`x & y  x | y  x ^ y`
`~ x`
`x << n  x >> n  x >>> n`
	`x mod n`
	`x land y  x lor y  x lxor y`
	`lnot x`
	`x lsl n  x asr n  x lsr n`

types algébriques	

`abstract class T { }`
`class Nil extends T { }`
`class Cons extends T {`
`  int head; T tail; }`
 	
`T l = new Cons(1,`
`  new Cons (2, new Nil()));`
 	
`abstract class T {`
`  abstract int length(); }`
`class Nil {`
`  int length() { return 0; } }`
`class Cons { int length() {`
`  return 1+tail.length(); } }`
	`type t =`
	`  | Nil`
	`  | Cons of int * t`
	
	`let l = Cons (1, Cons (2, Nil)) in`
	
	`let rec length l = match l with`
	`  | Nil -> 0`
	`  | Cons (_, r) -> 1 + length r`

exceptions	

`class E extends Exception { }`
`throw new E();`
`try { ... } catch (E e) { ... }`
	`exception E`
	`raise E`
	`try ... with E -> ...`

égalité	

`x == y`
(égalité physique)
 	
`x.equals(y)`
(égalité structurelle, si elle est programmée)
	`x == y`
	(même chose)
	
	`x = y`
	(égalité structurelle, fournie par OCaml)