J-O-Caml (1)

jean-jacques.levy@inria.fr

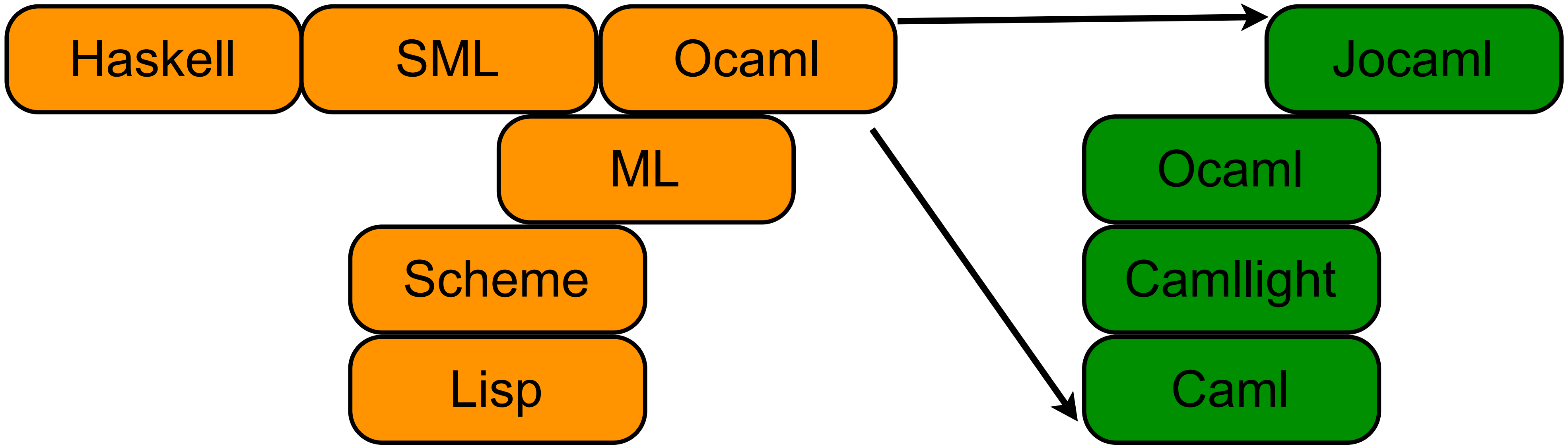
Qinghua, November 19

Plan of this class

- writing programs in Ocaml
- functional programming
- use of polymorphic types
- pattern-matching
- tour in the libraries

Functional programming

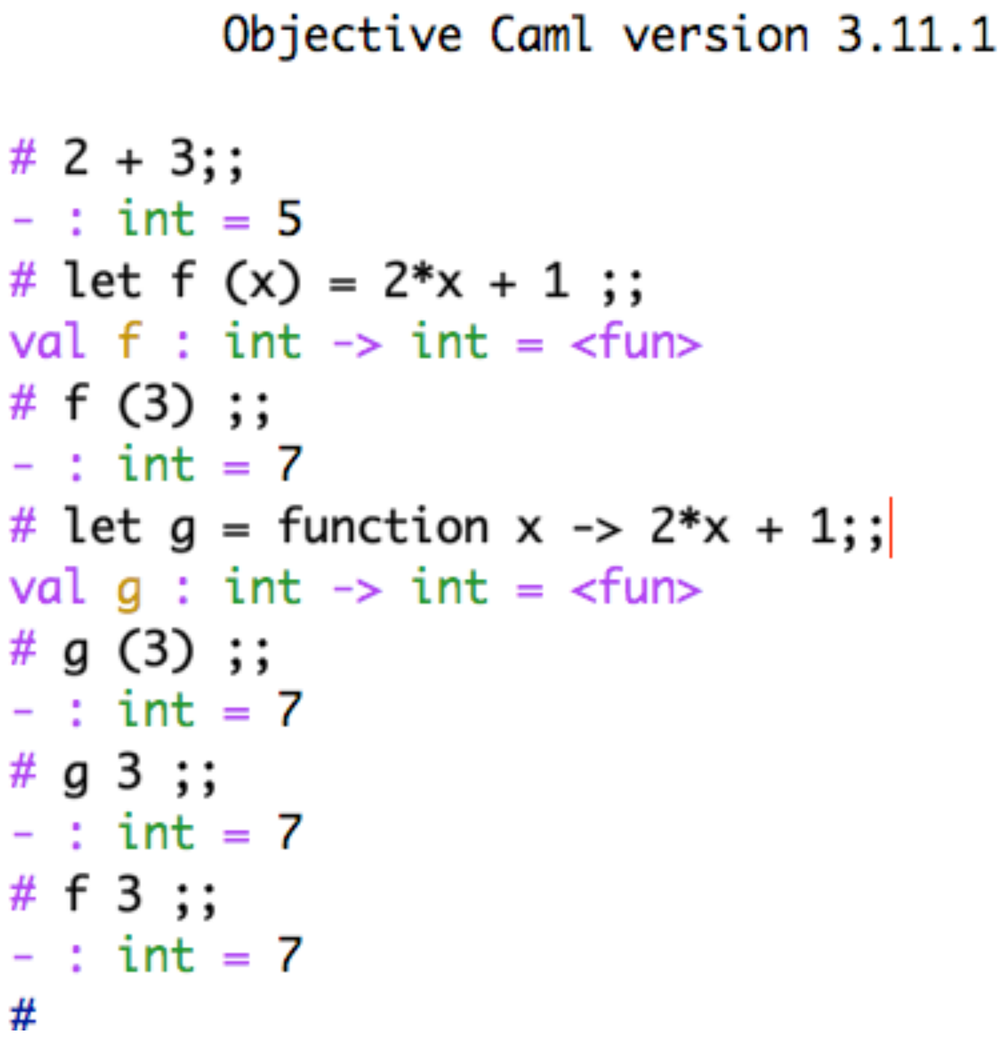
- Scheme, SML, Ocaml, Haskell are functional programming languages
- they manipulate functions
- and try to reduce memory states



Installing Ocaml

- google Ocaml
- `caml.inria.fr/index.en.html`
- download the system (Linux, MacOS, Windows)
- results in:
  - `ocaml` (interactive toplevel)
  - `ocamlc` (compiler)

Phrases at toplevel



```
        Objective Caml version 3.11.1

# 2 + 3;;
- : int = 5
# let f (x) = 2*x + 1 ;;
val f : int -> int = <fun>
# f (3) ;;
- : int = 7
# let g = function x -> 2*x + 1;;
val g : int -> int = <fun>
# g (3) ;;
- : int = 7
# g 3 ;;
- : int = 7
# f 3 ;;
- : int = 7
#
```

$$f(x) = 2 * x + 1$$

$$g = \lambda x.\, 2 * x + 1$$

$$f \equiv g$$

Scopes of definitions

- fine control in definition scopes

  `let`, `let in`, `let and`, `let and in`, `let rec`, `let rec in` ...

$\texttt{let}\, x = M\ \texttt{in}\ N$

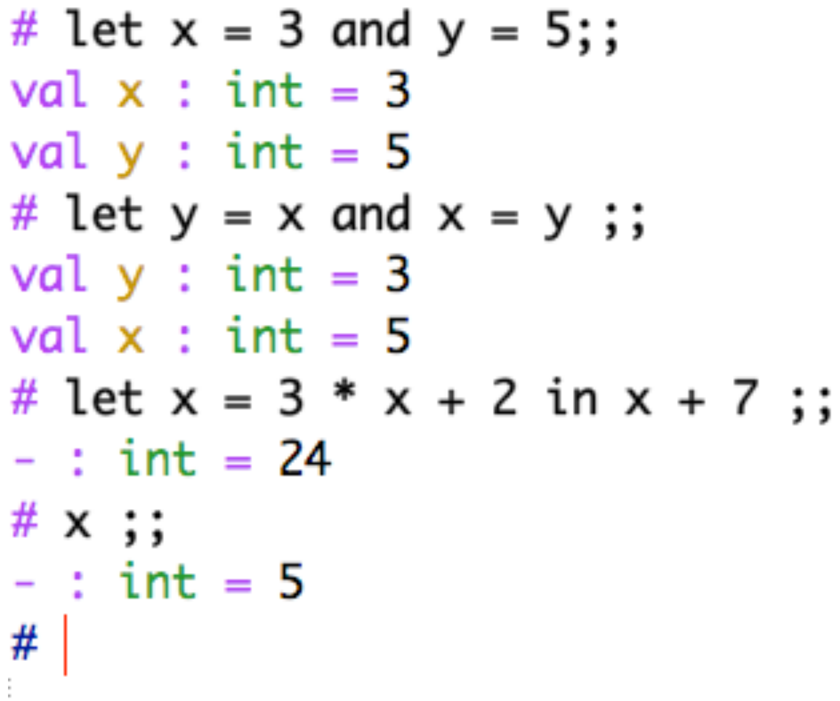
$\texttt{let}\, x = M\ ;;$

$\texttt{let}\, x_1 = M_1\ \texttt{and}\ x_2 = M_2\ \texttt{in}\ N$

$\texttt{let}\, x_1 = M_1\ \texttt{and}\ x_2 = M_2\ ;;$

$\texttt{let rec}\, x = M\ \texttt{in}\ N$

$\texttt{let rec}\, x = M\ ;;$



```
# let x = 3 and y = 5;;
val x : int = 3
val y : int = 5
# let y = x and x = y ;;
val y : int = 3
val x : int = 5
# let x = 3 * x + 2 in x + 7 ;;
- : int = 24
# x ;;
- : int = 5
#
```

Scopes of definitions

- fine control in definition scopes

  `let`, `let in`, `let and`, `let and in`, `let rec`, `let rec in` ...

$\texttt{let}\, x = M \;\texttt{in}\; N$

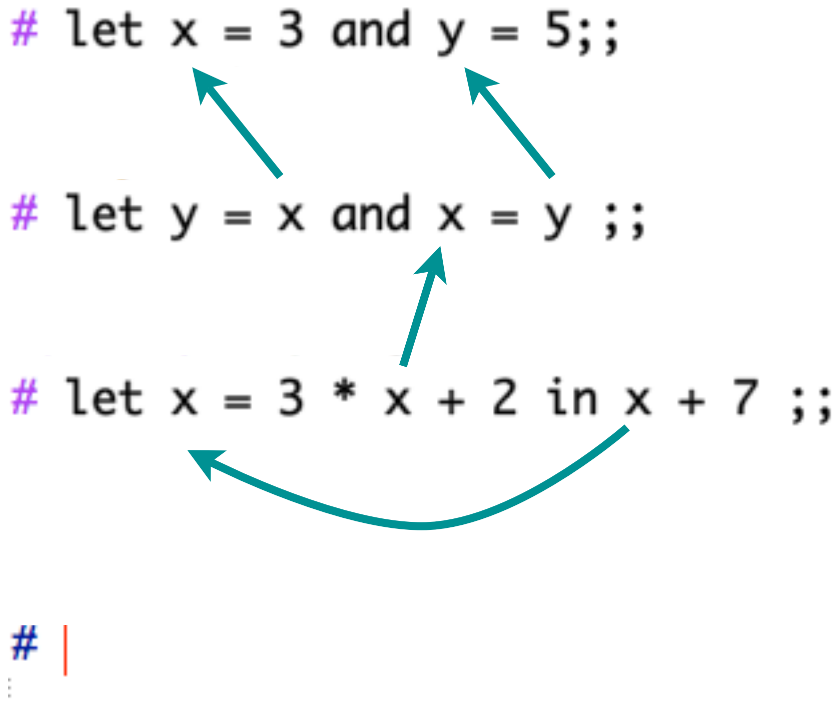
$\texttt{let}\, x_1 = M_1 \;\texttt{and}\; x_2 = M_2 \;\texttt{in}\; N$

$\texttt{let rec}\, x = M \;\texttt{in}\; N$

$\texttt{let}\, x = M \;\texttt{;;}$

$\texttt{let}\, x_1 = M_1 \;\texttt{and}\; x_2 = M_2 \;\texttt{;;}$

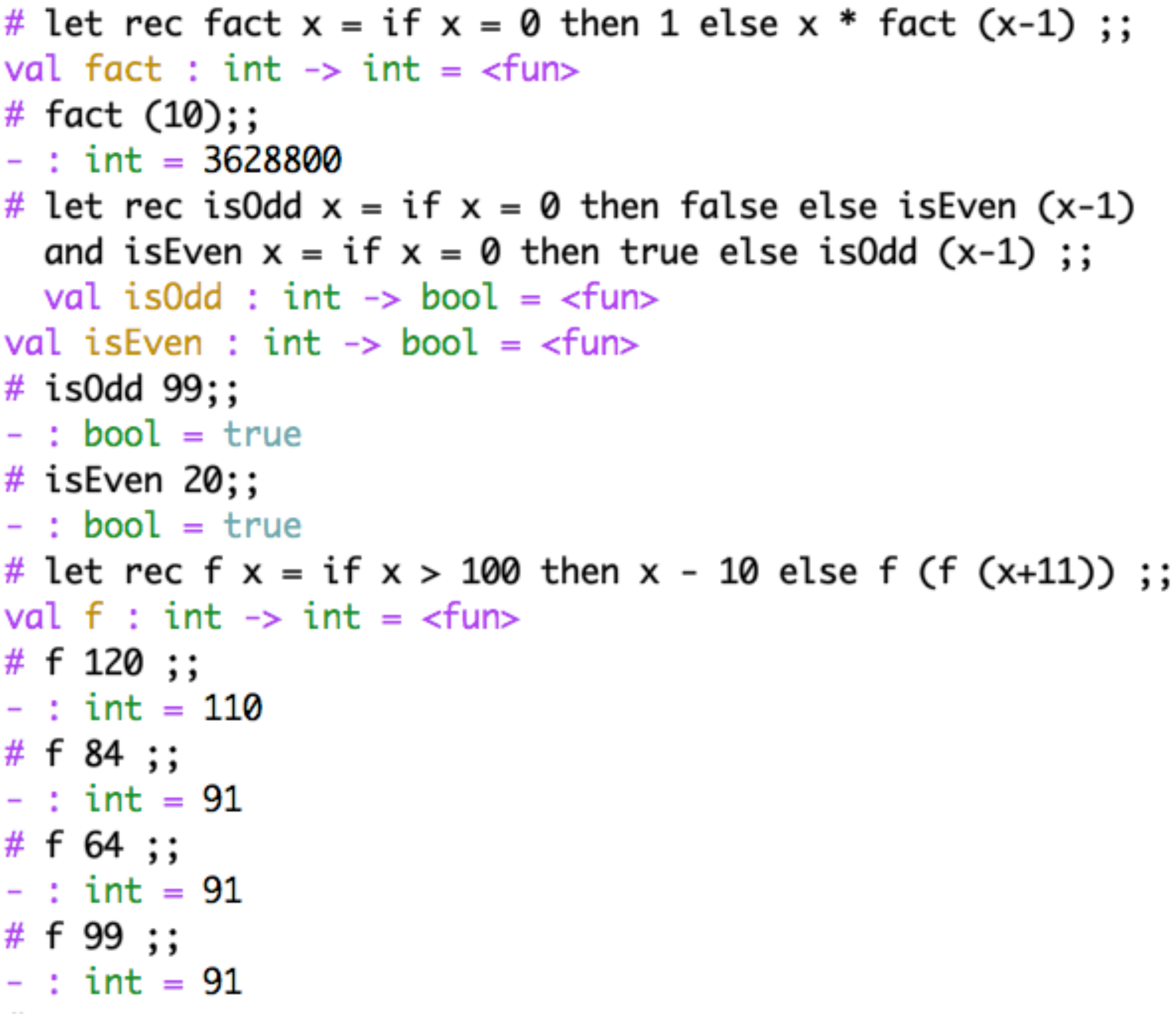
$\texttt{let rec}\, x = M \;\texttt{;;}$



```
# let x = 3 and y = 5;;

# let y = x and x = y ;;

# let x = 3 * x + 2 in x + 7 ;;

# |
```

Scopes of definitions

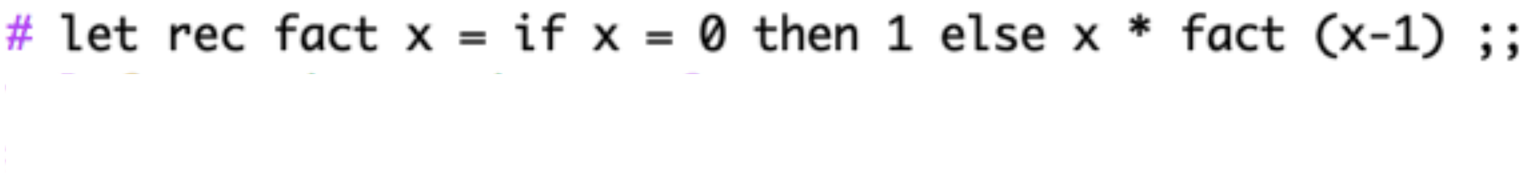
- fine control in definition scopes



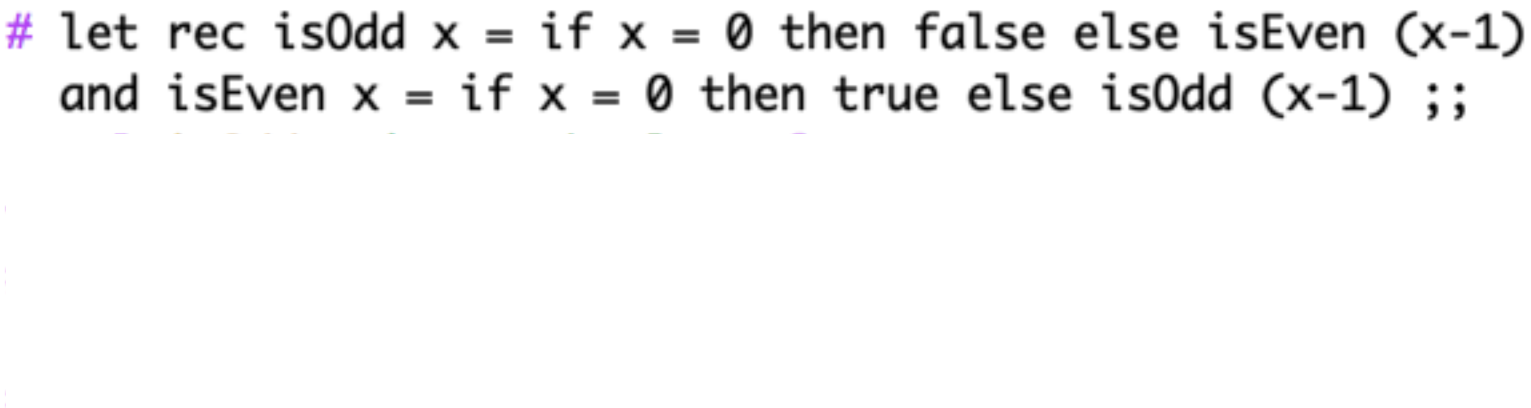
```
# let rec fact x = if x = 0 then 1 else x * fact (x-1) ;;
val fact : int -> int = <fun>
# fact (10);;
- : int = 3628800
# let rec isOdd x = if x = 0 then false else isEven (x-1)
  and isEven x = if x = 0 then true else isOdd (x-1) ;;
  val isOdd : int -> bool = <fun>
val isEven : int -> bool = <fun>
# isOdd 99;;
- : bool = true
# isEven 20;;
- : bool = true
# let rec f x = if x > 100 then x - 10 else f (f (x+11)) ;;
val f : int -> int = <fun>
# f 120 ;;
- : int = 110
# f 84 ;;
- : int = 91
# f 64 ;;
- : int = 91
# f 99 ;;
- : int = 91
```

Scopes of definitions

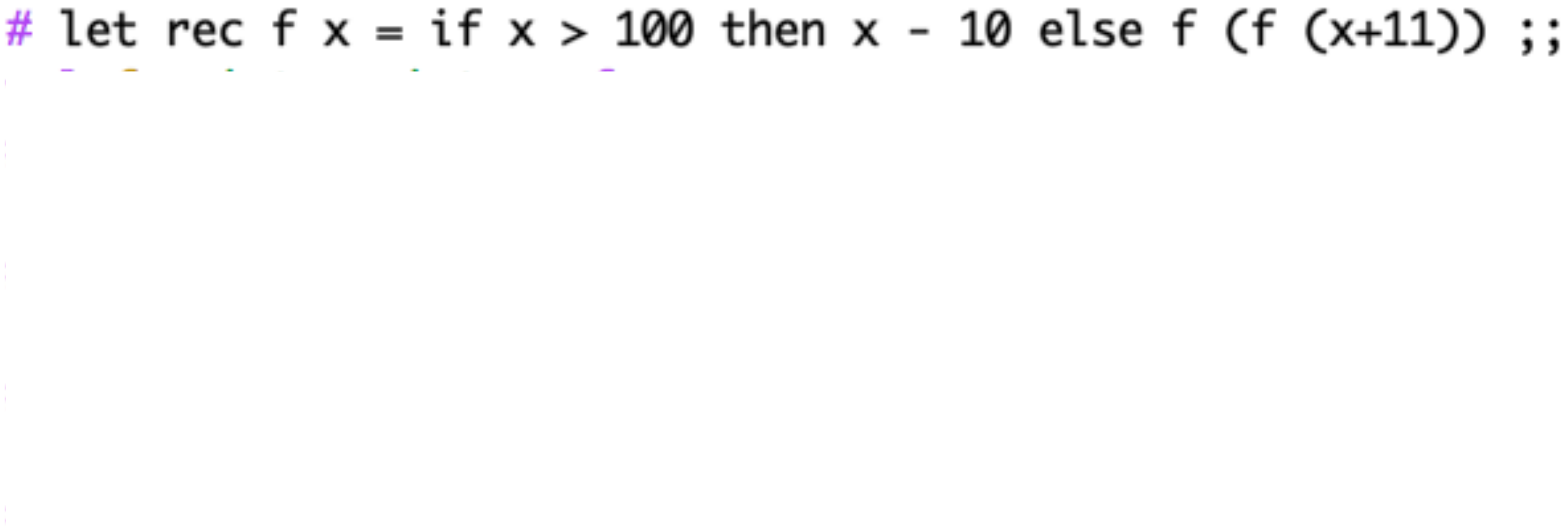
- fine control in definition scopes



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



```
# let rec isOdd x = if x = 0 then false else isEven (x-1)
  and isEven x = if x = 0 then true else isOdd (x-1) ;;
```



```
# let rec f x = if x > 100 then x - 10 else f (f (x+11)) ;;
```

Basic types

- `int` (integers) `1, 2, 3, …`
- `float` (real numbers) `2.3, 1.2, 0.`
- `char` (characters) `‘a’, ‘b’, ‘c’, …`
- `bool` (booleans) `true, false`
- `unit` (void) `()`

Compound built-in types

- `string` (strings) `“nihao”, …`
- `list` (lists of any type) `[1; 2], 3 :: [4; 6]`
- `array` (arrays of any type) `[| 1; 2; 3; 4 |]`

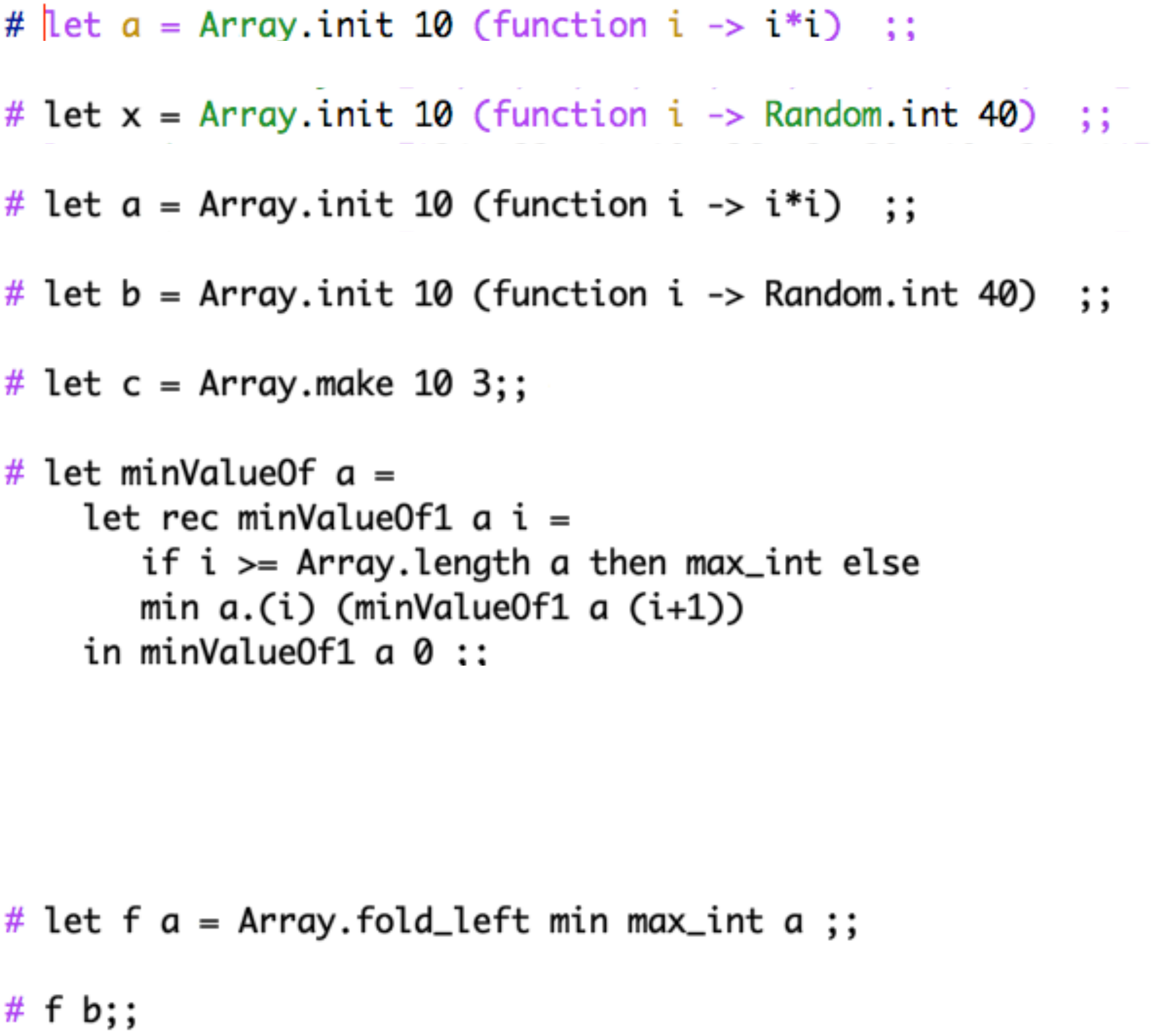
No overloading in Ocaml

- `3 + 4` (on integers)
- `4.5 +. 3.0` (on real numbers)
- `3 + 2.4` (not allowed)
- `(float_of_int 3) +. 2.4` (legal expression)
- this is to ease type inference

Operations on compound types

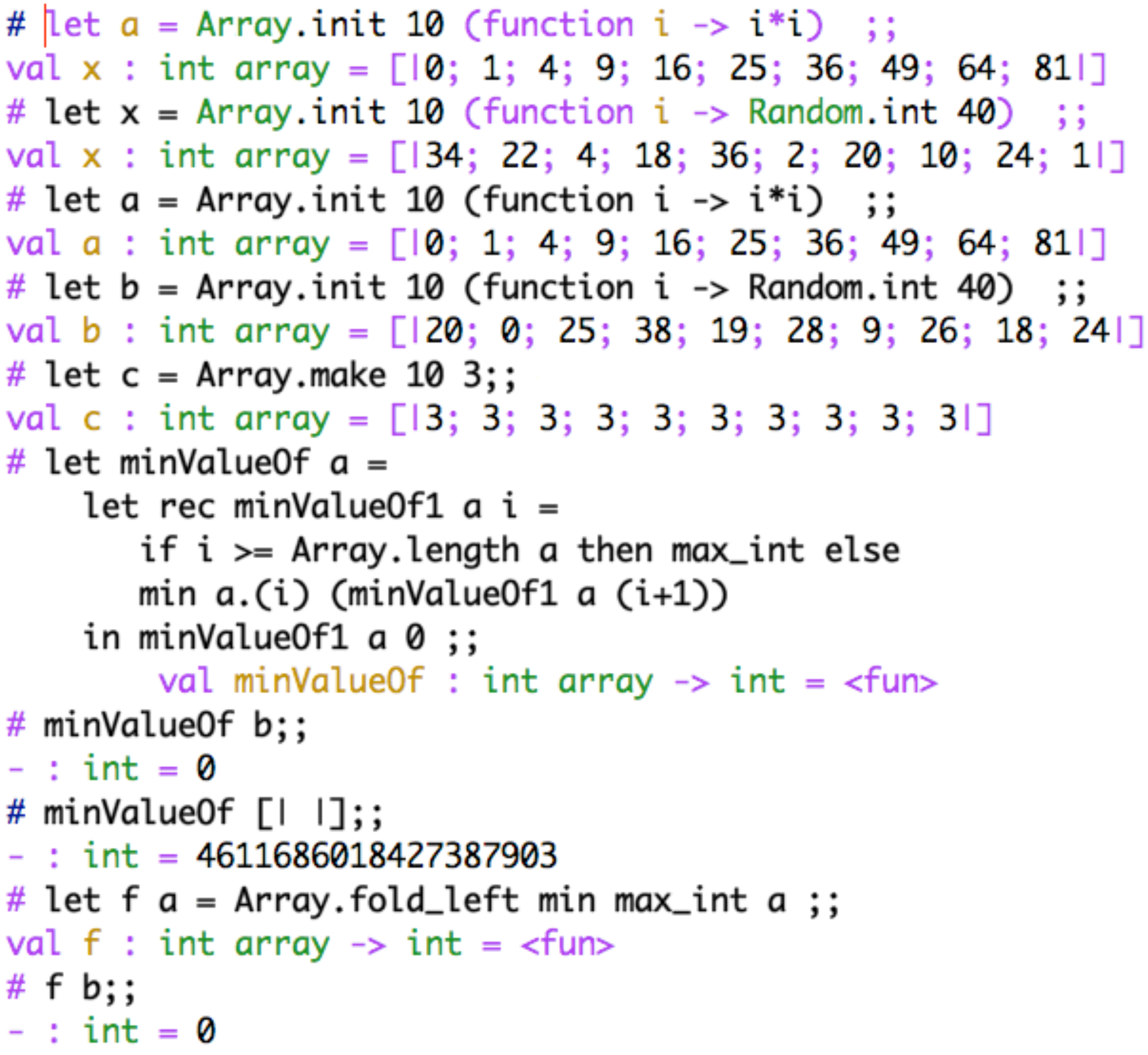
- `“nihao”.[3]` (character at position)
- `List.hd [1; 2], List.tl [1; 2]` (head and tail of list)
- `[| 1; 2; 3; 4 |].(3)` (element at some index in array)

Small examples on arrays



```
# let a = Array.init 10 (function i -> i*i)  ;;

# let x = Array.init 10 (function i -> Random.int 40)  ;;

# let a = Array.init 10 (function i -> i*i)  ;;

# let b = Array.init 10 (function i -> Random.int 40)  ;;

# let c = Array.make 10 3;;

# let minValueOf a =
    let rec minValueOf1 a i =
       if i >= Array.length a then max_int else
       min a.(i) (minValueOf1 a (i+1))
    in minValueOf1 a 0 ;;




# let f a = Array.fold_left min max_int a ;;

# f b;;
```

Small examples on arrays



```
# let a = Array.init 10 (function i -> i*i)  ;;
val x : int array = [|0; 1; 4; 9; 16; 25; 36; 49; 64; 81|]
# let x = Array.init 10 (function i -> Random.int 40)  ;;
val x : int array = [|34; 22; 4; 18; 36; 2; 20; 10; 24; 1|]
# let a = Array.init 10 (function i -> i*i)  ;;
val a : int array = [|0; 1; 4; 9; 16; 25; 36; 49; 64; 81|]
# let b = Array.init 10 (function i -> Random.int 40)  ;;
val b : int array = [|20; 0; 25; 38; 19; 28; 9; 26; 18; 24|]
# let c = Array.make 10 3;;
val c : int array = [|3; 3; 3; 3; 3; 3; 3; 3; 3; 3|]
# let minValueOf a =
    let rec minValueOf1 a i =
       if i >= Array.length a then max_int else
       min a.(i) (minValueOf1 a (i+1))
    in minValueOf1 a 0 ;;
        val minValueOf : int array -> int = <fun>
# minValueOf b;;
- : int = 0
# minValueOf [| |];;
- : int = 4611686018427387903
# let f a = Array.fold_left min max_int a ;;
val f : int array -> int = <fun>
# f b;;
- : int = 0
```

Easy exercices

- `isPalindromic s` returns true if `s` is palindrome
- `reverse s` returns mirror image of `s`

(`Use s.[i] <- c` store character `c` at (`i` + 1) position in `s`)

More exercices

- `sort a` sorts array `a` in place

  (Use `a.(i) <- x` store x at (i + 1) position in `a`)

- `transpose a` transposes matrix `a` in place

  (Use `a.(i).(j) <- x` store x at (i + 1), (j + 1) position in `a`)

  (also `Array.make_matrix h w v` creates `h x w` matrix filled with value `v`)

Objective for next classes

- a labeling algorithm for bitmap graphics

Combien d’objets dans une image?

Jean-Jacques Lévy
INRIA

CENTRE DE RECHERCHE COMMUN
INRIA
MICROSOFT RESEARCH

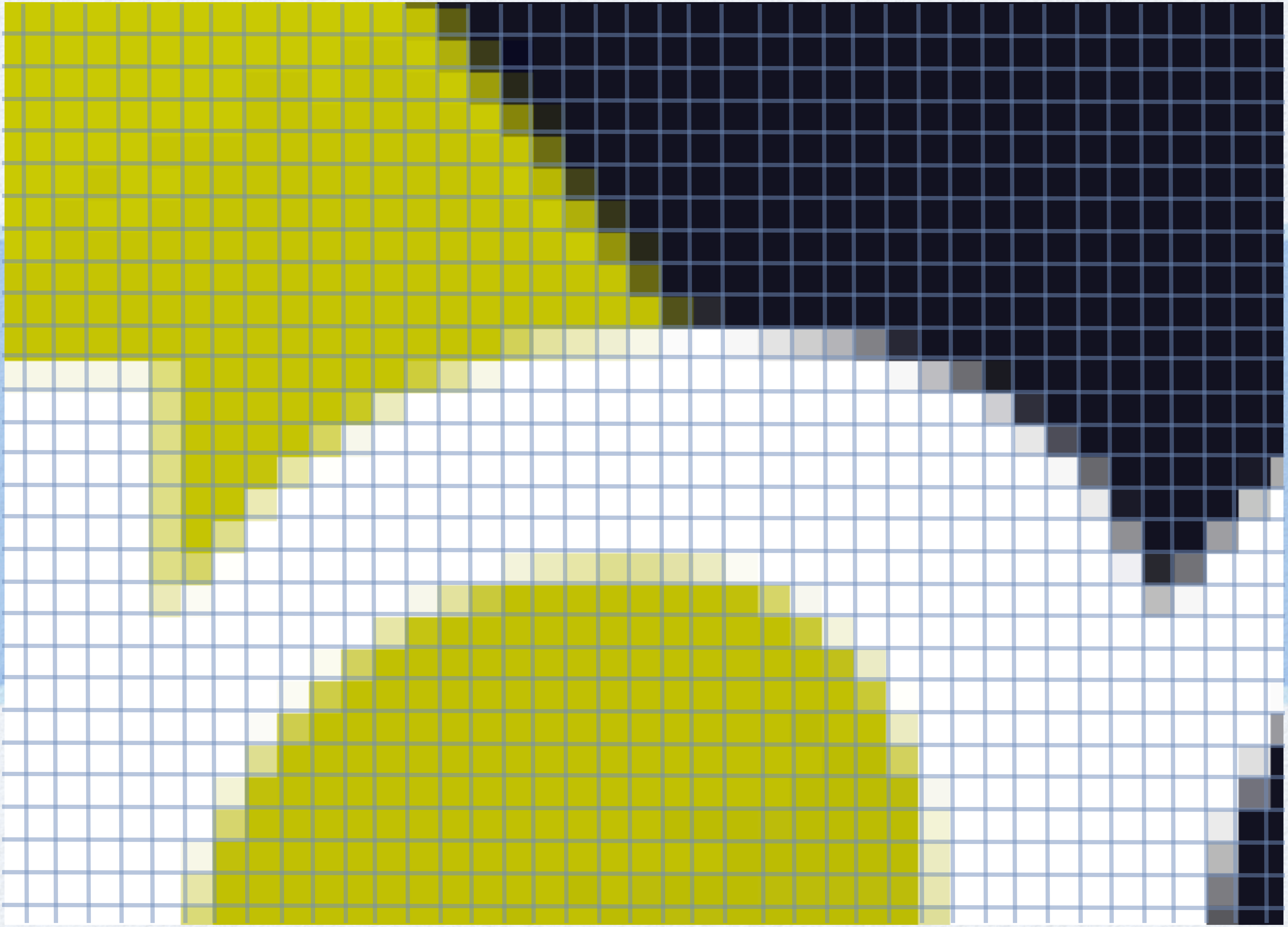
Combien d’objets dans une image?

Jean-Jacques Lévy
INRIA

CENTRE DE RECHERCHE COMMUN
INRIA
MICROSOFT RESEARCH

Combien
dans une

mb



PIXELS

(pictures elements)

Combien d'objets dans une image?

Jean-Jacques Lévy
INRIA

CENTRE DE RECHERCHE
COMMUN

INRIA
MICROSOFT RESEARCH

800

1200

10^6

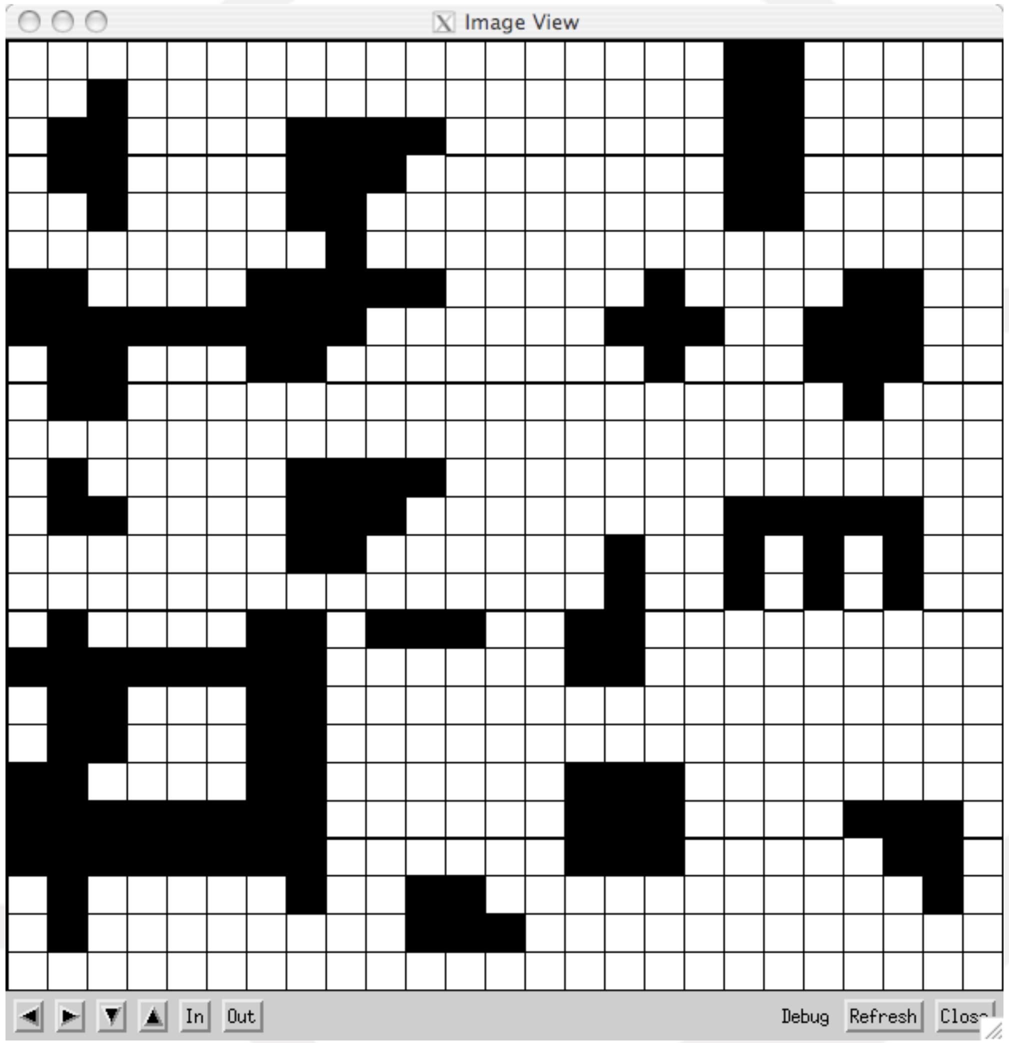
1Mpix

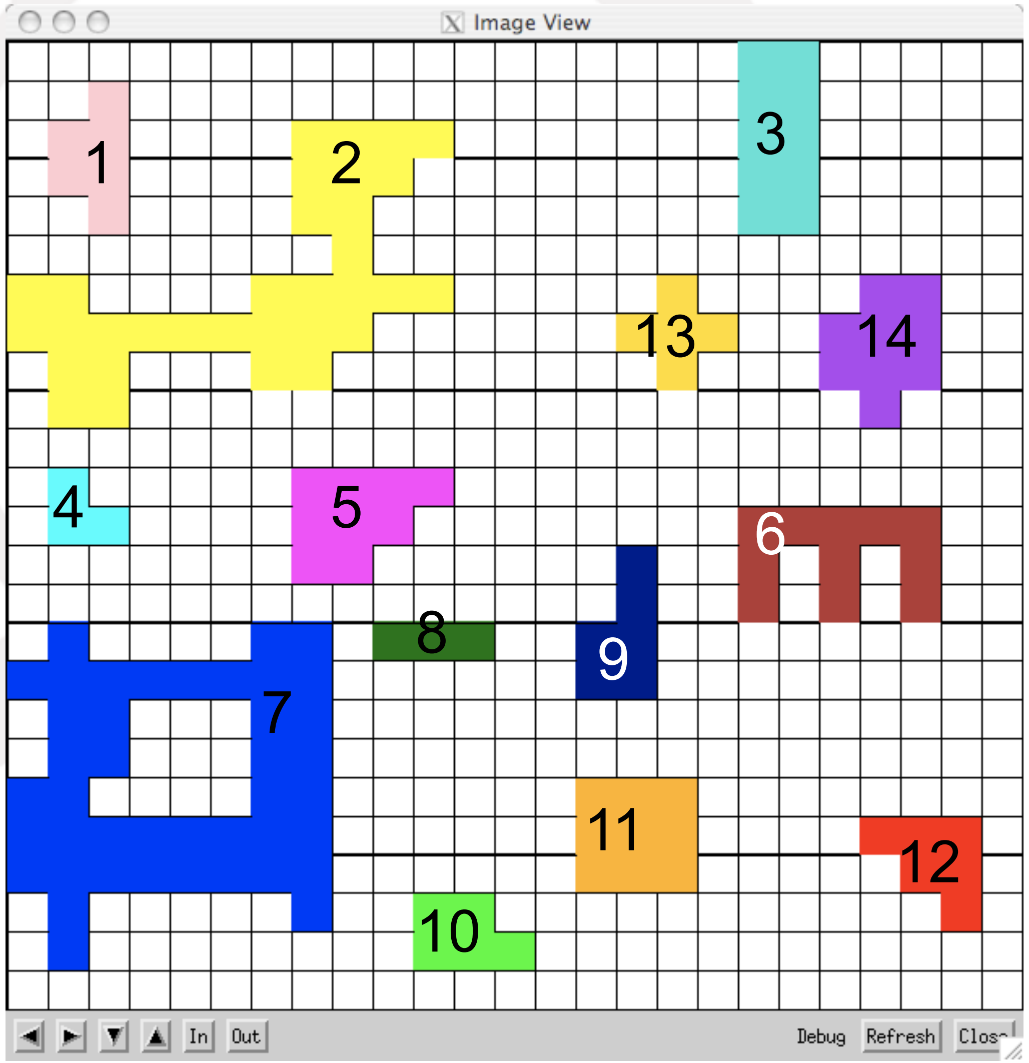
Problem and Algorithm

What is an object?

- set of similar adjacent pixels
  - similar ?
- simplification
  - grayscale images (255 values)
  - 0 = black, 255 = white
  - similar = adjacent with close values
- give a disctinct number to each object
- number of objects is max of previous numbers

Labeling





15 objects in this picture

Exercise for next class

- find an algorithm for the labeling algorithm