The OCaml MOOC

Benjamin Canou, Yann Régis-Gianas

(joint work with Çagdas Bozman, Roberto Di Cosmo, Grégoire Henry and Ralf Treinen)

Canterbury, June 22, 2017

Trends in Functional Programming in Education 2017

A humble and ambitious journey

Humble ?

- Only **3 key notions** of languages from the ML family :
  - Functional programming,
  - Static typing with full type inference, and
  - Algebraic datatypes.
- Write **medium sized** programs of **reasonable complexity** using OCaml.

…and Ambitious ?

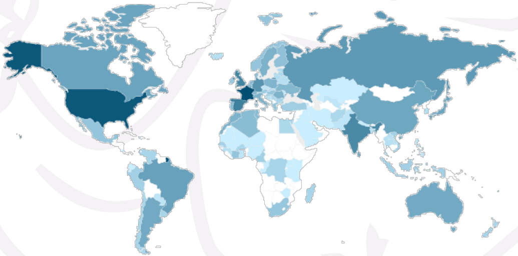
— Ensure that the learners who completed the course **master the 3 key notions**.
— …in just **7** weeks.

A secret goal :

Retain them all along the course.

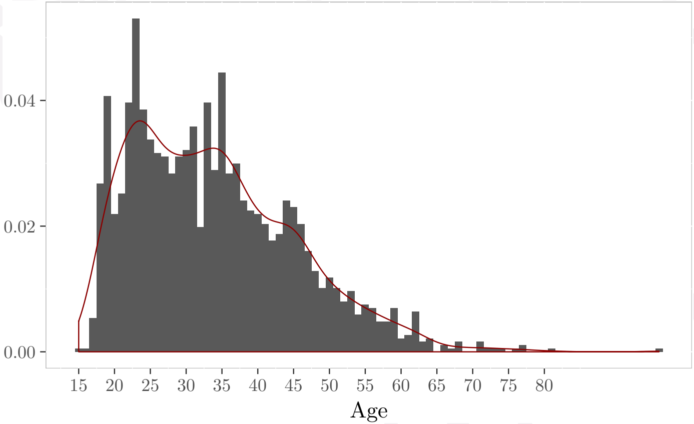
Who were our students ?

Where are they from?



— **7000** learners enrolled;
— from more than **120** countries
(France, US, Spain, UK, India and Germany being the top 6);
— **2418** actually showed up when the course started.

How old are they ?



How educated are they ?

	Active		Passed	
	Numbers	avg. success	Numbers	avg. success
High School	255	20.93	32	90.19
Associate	45	21.29	4	86.50
Bachelor	567	26.70	113	87.12
Master	775	37.13	242	86.69
PhD	226	38.00	75	91.44
Elementary	7	26.43	1	100.0
Junior High	65	21.14	9	85.44
Other	40	34.02	11	86.45
N/A	438	34.93	128	89.41
Total	2418		615	

Unusual for a MOOC : a lot of students.

How did our students learn ?

Study and PRACTICE!

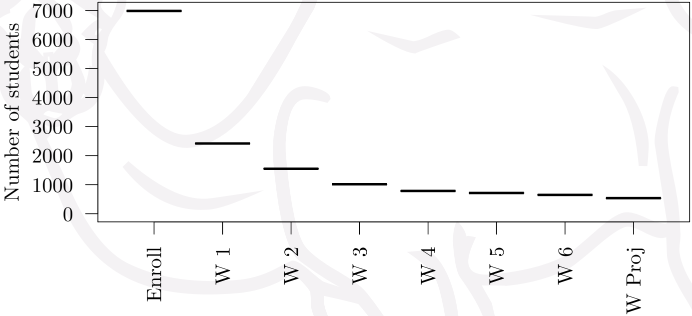
- Classic material : slides and videos (42 capsules, ∼6 hours);
- State-of-the-art exercise environment;
- 55 exercises (7 quizz, 48 automatically graded exercices) and 2 projects.
- The total length of our own solutions is around 1500 loc.
- One important pedagogical idea :

No submission limit and deadlines on the exercises

- The consequence :

Students are working harder to get a score of 100%

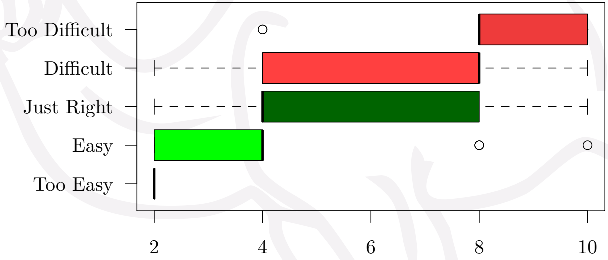
How good was the trip ?



After week 2, we had gathered a stable group of students

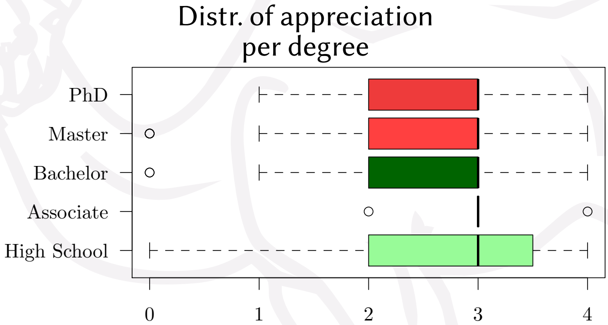
How hard was it for the learners?

Weekly effort (hs/week)
per perceived difficulty



A majority of students found the difficulty just right
and spend 8 hours per week on the course!

How pleasant was the trip ?



Whatever the level of education, our students seem satisfied.

The Exercise Platform

A beginner's IDE in the browser

Main features

— everything runs inside the browser, nothing to install;
— syntax colouring and forced indentation;
— OCaml runs inside a separate worker for responsiveness;
— incremental, randomized automated grading system;
— structured report with pretty printed test cases;
— interactive toplevel for quick testing.

We are building a standalone platform from this code : learn-ocaml.

DEMO TIME

A minimal grader

Exercise text

```
<p>Given the type of binary tree in the
 prelude, write the following function</p>
<ol>
  <li>
    <code>size: tree -> int</code>
    computes the number of tree elements.
  </li>
  <li>
    <code>height: tree -> int</code>
    computes the height of the tree;
  </li>
</ol>
```

A minimal grader

Prelude

```
type tree =
  | Empty
  | Node of tree * int * tree
```

Template

```
let rec size = "Replace␣by␣your␣code." ;;
let rec height = "Replace␣by␣your␣code." ;;
```

Solution

```
let rec size = function
| Empty -> 0
| Node (l,_,r) -> 1 + size l + size r ;;
let rec height = function
| Empty -> 0
| Node (l,_,r) ->
    1 + max (height l) (height r) ;;
```

A minimal grader

Grader

```
let sample_tree () =
  let rec aux lvl =
    match Random.int lvl with
    | 0 -> Empty
    | _ -> Node (aux (lvl-1), Random.int 100, aux (lvl-1))
  in aux 4 ;;
set_result @@ ast_sanity_check code_ast @@ fun () ->
  [ Section
     ([ Text "Exercise␣1:␣" ; Code "size"],
        test_function_1_against_solution
          [%ty: tree -> int] "size" []) ;
    Section
     ([ Text "Exercise␣2:␣" ; Code "height"],
        test_function_1_against_solution
          [%ty: tree -> int] "height" []) ] ;;
```

Graders are typed, with local run-time checks where student code is loaded.

How deep can we probe student code?

Not only the output!

- syntax (graders can insert AST rewriting steps)
  (e.g. write this function using a single `match` expression);
- variable definition and scoping (using the typed AST)
  (e.g. define this value using at most n local variables);
- function definition and application (including partial ones)
  (e.g. perform this effect after passing the 3rd argument);
- side effects by introspecting references in student code
  (e.g. randomize this array in place);
- complexity evaluation by instrumenting the standard operators
  (e.g. sort an array using at most $n \cdot \log(n)$ accesses);
- modules, interfaces and ADTs (turned into module packages by the grader)
  (e.g. implement this interface for functional tries).

How deep can we probe student code ?

A successful approach :

— fair grader size / solution size ratio ;
— good confidence thanks to static typing of graders ;
— authors can concentrate on generating good test cases.

Technical limitations :

— command line toolchain and separate compilation ;
— system libraries and I/Os (all inside the browser) ;
— synchronous interaction (must conform to the event loop).

Main critics :

— students don't write their own types (before ADTs are introduced) ;
— no feedback on the style of the code.

Conclusion

- Let the students try again, and again, and again.
- Automatic grading is appreciated by students!
- We have trained **600** fresh camlers, ready to contribute to the community!

The OCaml MOOC

Thank you !

List of projects and exercises

A Solver for Klotski
Random Text Generation
Integer identifiers
String identifiers
Simple functions over integers
Simple functions over strings
Prime numbers
Enigma
Points and vectors
Time on planet Shadokus
Finding the minimum
Searching for strings in arrays
A small typed database
Pattern matching exhaustivity
A type for array indexes
The option type
First In First Out
Classic functions over lists
Symbolic manipulation of arithmetic expressions
Tries

Type directed programming
Balanced binary trees
List with an efficient concatenation
Advanced patterns
Lambda
Using first class functions
Functions returning functions
Optimizing partial applications
A small arithmetic interpreter
Using and writing the map function
Using fold to produce lists
Using fold to check predicates
Optimising a tree traversal using exceptions
Unraveling the automatic grader
Printing lists

Displaying a Filesystem Hierarchy
Printing with loops
Producing fine ASCII art
Rotating the contents of an array
Implementing a stack with an array
Implementing mutable lists
Simple uses of references
Reading lines from the standard input
Opening modules
Accessing modules and submodules
Wrapping functions in a module
Type abstraction using a signature
Multiset
Remove elements from dictionaries
Char indexed hashtables