Learn OCaml

An Online Learning Center for OCaml

Benjamin Canou
OCamlPro, Paris
benjamin.canou@ocamlpro.com

Grégoire Henry
OCamlPro, Paris
gregoire.henry@ocamlpro.com

Çağdaş Bozman
OCamlPro, Paris
cagdas.bozman@ocamlpro.com

Fabrice Le Fessant
INRIA Paris-Rocquencourt
fabrice.le_fessant@inria.fr

Abstract

We present Learn OCaml, a Web application that packs a set of learning activities for people who want to learn OCaml. It includes an integrated and reworked version of the venerable Try OCaml, and an exercise environment with automated grading derived from the one developed for the OCaml MOOC. It works entirely in the browser, the server being used for storing static files and synchronizing between different devices. A special effort has been made to make it usable on tablets, and even mobiles. A main public instance will be hosted at OCamlPro, but the project is open-source, and universities can host their own version on site. We will also provide a public repository for teachers to contribute lessons and exercises.

A demo is currently hosted at: http://try.ocamlpro.com/learn-ocaml-demo.
The source code is available as free software at: https://github.com/ocamlpro/learn-ocaml.

1 Introduction

Learn OCaml is a new Web platform than combines several learning tools. It is targeted at a broad audience: at curious people to get a quick taste of OCaml, at newcomers to learn the language, at casual users to refresh or improve their skills, and at regular OCaml programmers to learn about and try new features.

Building on several past experiments by OCamlPro, it currently features a sophisticated toplevel, an interface for browsing quick tutorials and another for reading longer lessons, and an environment for solving exercises with automatic grading. We are open to other features proposals, or even contributions, as the platform is free software placed under the GNU Affero GPL.

The next three sections present the main learning activities that have already been developed, giving for each a quick description, a bit of background, and some example uses. The last section discusses our (open) ambitions for this project. Several commented views of the interface are given in appendix.

2 The Toplevel Activity

A first learning activity, also used as a component in the other activities, is an OCaml toplevel. This toplevel introduces several new features, compared to the demonstration shipped with js_of_ocaml or the one of Try OCaml.

To run OCaml code, we use a bytecode executable that runs a custom toplevel loop. We compile it to JavaScript using the toplevel-compatible compiling mode of js_of_ocaml.

We wrote a richer version of the Toploop module that allows us to trap the various outputs of the compiler. This way, we can present them to the user using different styles for better readability, colorize code and toplevel answers, and highlight the locations of warnings and errors.

The toplevel loop also traps the standard output channels of the evaluated code, so that it may prevent display flooding. After a piece of code exceeds some predefined amount of output, it is buffered and retained until the user selects to either drop or show it from a dialog box.

A similar protection is also provided against infinite loops. When the run time of a phrase exceeds some given number of seconds, the user is asked if she wants to give it a few more. Otherwise, it is killed after a countdown expires (so that the user does not let some hidden tab consume CPU time).

Figure 2 in the appendix shows several of these features.

For several reasons, including the flooding and looping prevention mechanisms, the toplevel does not run directly in the page, but in a separate process, using the standard Web Worker API. This choice has the important advantage of making the interface reactive even when code is running. But this isolation prevents the code to interact with the user directly. For that, the toplevel environment is enriched with callbacks to display raw HTML fragments, sent from the webworker, in the toplevel console. It would be possible, for instance, to implement a version of Graphics over this mechanism.

Also thanks to this structure, it could be fairly easy to run toplevel sessions on a remote server, for instance to target low power machines. It could also be possible to package Learn OCaml in an application, and then to use a standard (bytecode or native) toplevel executable.

3 The Tutorial Activity

Learn OCaml includes a new implementation of Try OCaml (an interactive OCaml tutorial in the browser developed by OCamlPro) that uses the new toplevel. Try OCaml lets the user choose a tutorial from a list. Each tutorial consists in a series of screens, containing some OCaml phrases with explanations and comments. The user can click on each phrase to see it interpreted in the side toplevel, and she can also try her own phrases.

This interface is not meant to replace a book. It is designed to present small tutorials, small chunk by small chunk. Currently, we use it for two main use cases. A first set of short tutorials covers the core features of OCaml, and serve as a quick tour of the language for curious users of other programming languages. A second set of tutorials is targeted at OCaml programmers, and cover the changes introduced by new releases of OCaml.

One of the difference with the previous implementation is that it is designed using the *responsive* features of modern browsers. Its interface will adapt itself automatically to the screen size and device capabilities of the machine on which it runs.

As seen in Figure 3 in the appendix, in the desktop version, the screen is split between the different parts. In the mobile version, some parts require a touch to be shown or hidden.

4 The Exercise Activity

One of the key elements of the OCaml MOOC, which really stood out compared to other programming MOOCs according to our end of MOOC survey, was its exercise environment. First, students did not have to install anything to enjoy learning OCaml, since the environment runs entirely inside the Web browser. So the effort needed to go from just watching the first videos to doing the first exercises was no more than a click. Once attracted, we observed in our statistics that almost all of the learners who completed the first exercises were immediately addicted to the system, and stayed until he end. Most of them even tried to get 100 points on all the exercises, even the optional ones. This is explained by the automated grader, which also runs inside the browser, and is thus able to give quick feedback. Solving an exercise becomes a game of incrementally satisfying the tests performed by the grader, until the report is all green. It was, actually, the starting point of the development of Learn OCaml, a platform to allow people to learn OCaml in such an entertaining fashion, out of the constraints of a MOOC.

Figure 1 in appendix shows the exercise environment in both mobile and desktop sizes. The student is first presented a list of exercises to choose from, which also indicates the status of the exercise (percentage of completion). This information is stored in the local storage of the browser, and can optionally be synchronized with a server. When she selects an exercise, she is given a tabbed interface, with a tab containing the text of the exercise, a tab with an OCaml editor to write her solution, a tab to display the grading report, and a toplevel initialized with the exercise prelude.

The text editor is based on the ACE JavaScript component. We developed an OCaml mode for it that uses `ocp-indent` to bring both syntax coloring and indentation. Indentation is mandatorily performed, in almost real time. This way, newcomers learn the OCaml formatting style without even thinking about it. This is an experimental behaviour that we tried on the MOOC learners, which turned out successful so we kept it in Learn OCaml.

The grading mechanism has been developed especially for this environment. It has been described in [1] (in French, an English article is in preparation). It would be far too long and off scope to present the details, but here are the main design choices. The grading mechanism is strongly tied to OCaml, which allows the grader to manipulate the values and types of the learner's code directly in a type-safe manner. It has been tested and refined on the 50 exercises of the OCaml MOOC, covering most of the functional and imperative features of OCaml, and an introduction to modules and functors. Graders are reasonably terse (around 3 times the size of solutions), and the type-aware design give great confidence in their strength. Once a grader passes the offline grader simulator on a reference solution, the teacher knows that the only things that may remain to be fixed are the exercise text and the random sampling bias.

5 Conclusion

We put lots of effort in developing this platform. The goal is to make the best Web learning center possible about OCaml. For that, lots of work remain, to improve and polish the user experience.

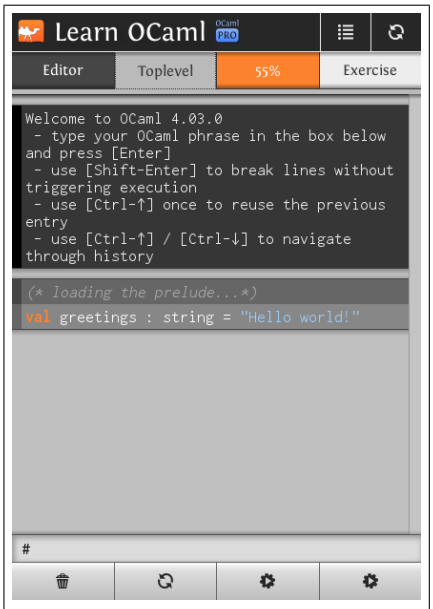
An important matter is the one of pedagogical content. For this, we will open a public repository for exercises and tutorials, that will be initially populated with contents from the MOOC and OCamlPro's own material. This talk will be the opportunity to gather volunteering and advice on that matter, especially from people with experience in teaching OCaml.

At the time of this talk, the code should be hosted on Github, distributed under an Affero GPL license. A main public instance of Learn OCaml will be hosted at OCamlPro, but universities and schools will be able to install the open-source project locally to host their own version of the site. We will also provide a public repository for teachers to share lessons and exercices, although it is not yet clear whether how to prevent students from accessing the solutions of the contributed exercices.

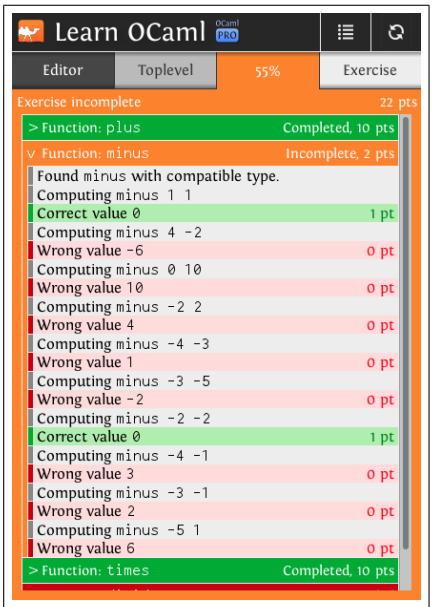
References

[1] Benjamin Canou, Çagdas Bozman, and Grégoire Henry. Sous le capot du mooc ocaml. In *Journées Francophones des Langages Applicatifs (JFLA) 2016*.

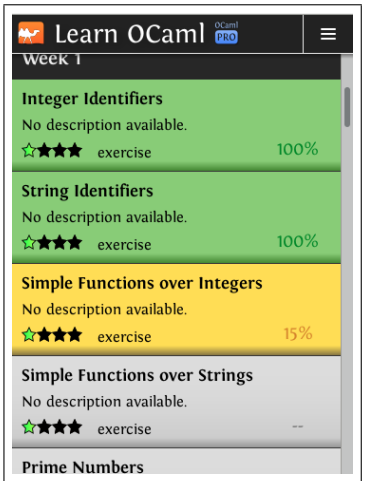
Appendix: Screen Captures



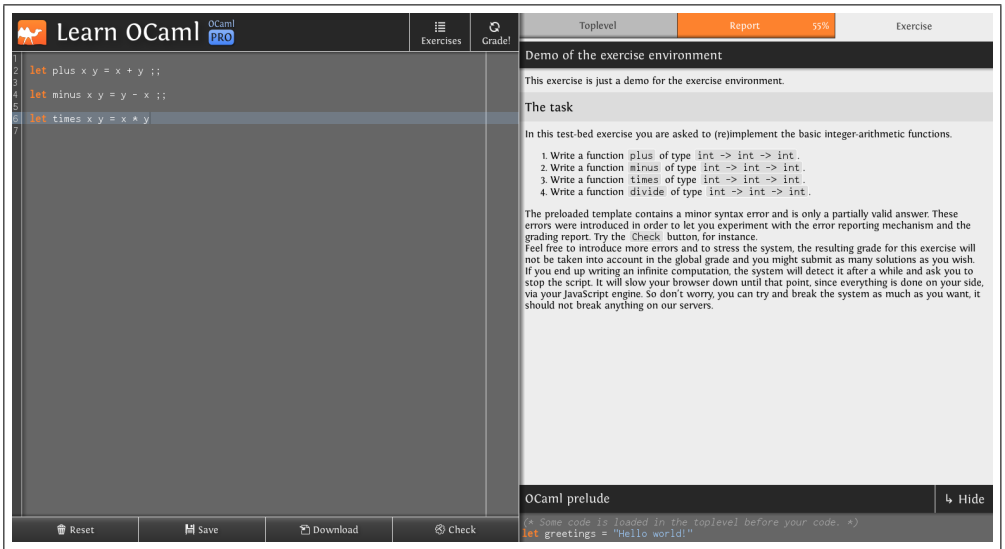
The toplevel tab, which is loaded with the exercise prelude.



The report tab, whose section are smartly unfolded to show the first failed test.

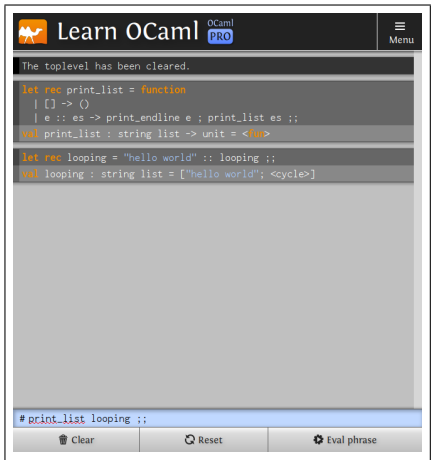


The list of exercises students can brag about when all green.

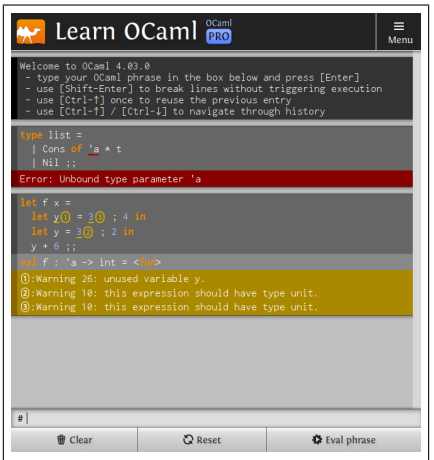


The desktop version always shows the editor pulled out of its tab on the left. Here, the exercise text tab is shown.

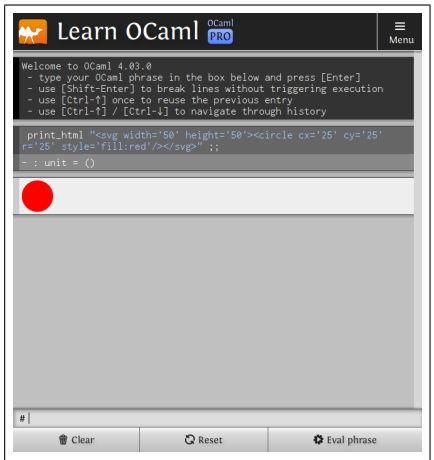
Figure 1: The exercise environment interface (mobile and desktop version).



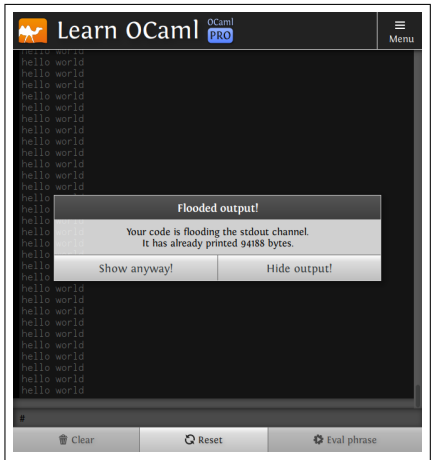
Toplevel phrases are separated, different styles help identify the different kinds of output.



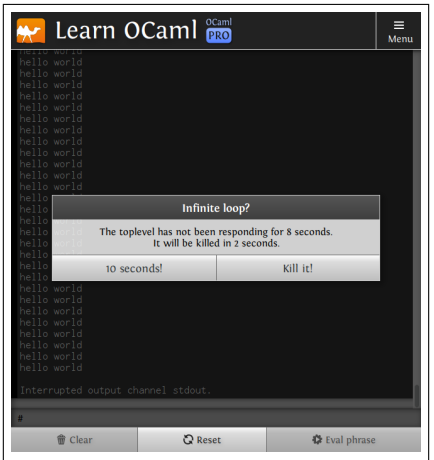
Locations in errors and warnings are numbered, the corresponding code is underlined.



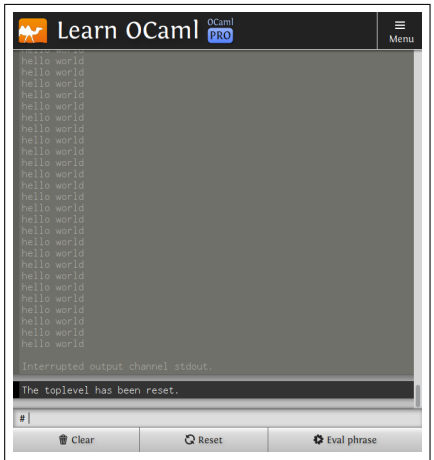
For advanced examples, is is possible to output HTML or SVG in a console block.



The toplevel detects a flood on stdout and asks the user if she wants to hide or display it.



If a run is too long, the user is asked for a few more seconds. Without answer, the toplevel is killed.



When interrupted, the toplevel is restarted. The user can retry its phrase by accessing the history.

Figure 2: The toplevel interface (tablet size)

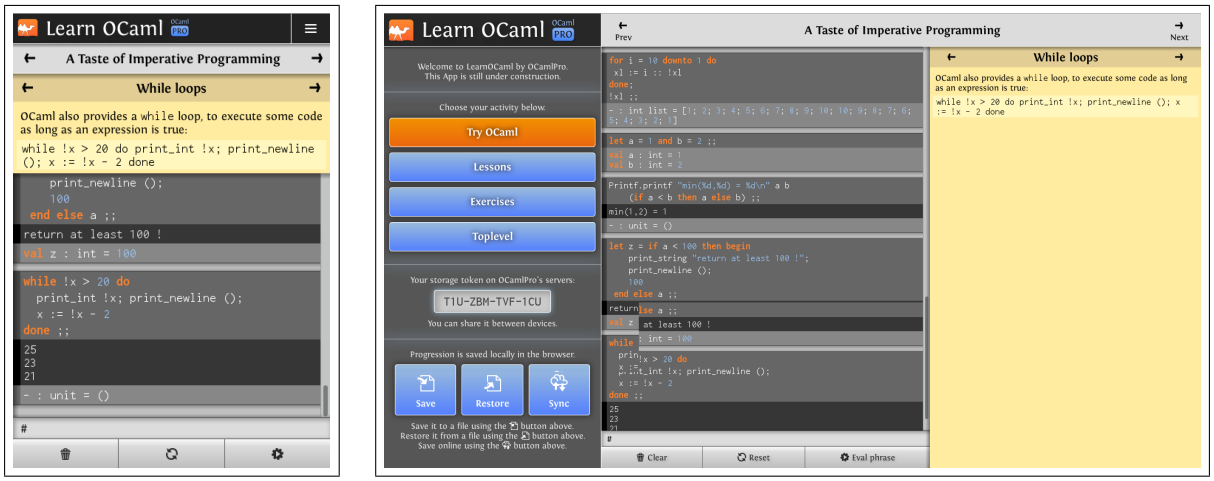


Figure 3: The Try OCaml interface (mobile and desktop versions).