CS Circles: An In-Browser Python Course for Beginners

David Pritchard
Department of Computer Science*
Princeton University
Princeton, NJ, USA
dp6@cs.princeton.edu

Troy Vasiga
David R. Cheriton School of Computer Science*
University of Waterloo
Waterloo, ON, Canada
tmjvasiga@cs.uwaterloo.ca

ABSTRACT

Computer Science Circles is a free introductory programming course in the form of a website. Its target audience is beginners, and its design goals are to be easy to use, fun, and to reach the broadest possible audience. We teach Python because it is simple yet powerful. The course content is structured but written in plain language. There are over 100 exercises embedded within its thirty lesson pages. This paper discusses the philosophy behind the course and its design, as well as a description of how it was implemented and statistics on its use.

Categories and Subject Descriptors

D.2.8 [**Computers and Education**]: Computer and Information Science Education

General Terms

Design, Human Factors, Languages

Keywords

Massive Online Open Courses, Python, CS0/1

1. INTRODUCTION

The last decade has seen an explosion of web-based software that runs "in the cloud," jointly with a tremendous improvement in the ability of a web browser to interact with users and servers. Near the end of 2009, we and our colleagues thought it would make sense to make an introductory programming course available in this way. Part of the motivation is that many wonderful online programming judges exist for veteran programmers[1], but there is little for

---

*CS Circles is a project of the Centre for Education in Mathematics and Computing, University of Waterloo

[1] e.g. TopCoder.com, Sphere Online Judge (`spoj.pj`), the ACM judge (`acm.uva.es`), USA Computing Olympiad training (`train.usaco.org`), the PEG judge (`wcipeg.com`), etc

beginners. It is unreasonable to expect the average beginner to download a compiler, or to re-load a web page and re-upload a file every time they want to test their "Hello, world!" program. Computer Science Circles is our way of filling this gap.

CS Circles is publicly available at

http://cscircles.cemc.uwaterloo.ca

Since opening in the fall of 2011, over 4000 users have registered and in total they have completed 90000 exercises. On an average day, the website handles 1000 visits and 7500 code submissions. Approximately 50% of the traffic is from users that are not logged in. The statistics in this paper are taken from the start of September 2012.

We eschew the traditional separation into "lesson/lecture content" and "problem sets." Our position is that such a framework puts the student at risk of losing interest during the lesson, and also splits their attention between review, reading exercises, and writing solutions when they finally start working hands-on. We think it is pedagogically superior for the exercises to be embedded in the lessons themselves: this reduces monotony and keeps the student on a focused progression. Removing page reloads and handling the student/auto-grader communication with asynchronous JavaScript (a.k.a. ajax) helps keep this experience smooth.

Programming is the ideal topic for computer-based learning: beginners can get immediate detailed feedback (from the compiler and auto-grader) on the entirety of their creations. Automatic grading is not a replacement for a real teacher looking on over your shoulder. Additionally, CS Circles does not check for plagiarism or style. To compensate for these issues we have added a human touch by letting students easily send a "Help" message when they are stuck. Any message sent in this way will also automatically include the code currently in the student's code editor for that problem. We have answered over 500 student questions in this way, with most messages answered in a 24-48 hour turnaround. Additionally, teachers wishing to use our site can be set as the "guru" of their students, which means all questions from their students will be sent to this guru instead; the guru has access to other student tracking tools as well.

The rest of the paper is organized as follows. Section 1.1 discusses related work. Section 2 provides an overview of the CS Circles experience and the philosophy behind it. Section 3 discusses the features of the site, grouped according to the three types of people interacting with the site: students, lesson authors, and teachers. Section 4 describes details of the site's implementation. Finally, Section 5 has some suggestions for future work.

1.1 Related Work

In the past year, the acronym MOOC became well-known in education, meaning "massive open online course." The two most well-known companies offering MOOCs are Coursera (coursera.org) and Udacity (udacity.com). Two main distinctions are that Coursera offers each course on a specific schedule, while Udacity allows open enrollment at any time; and that Coursera deals with the full range of university subjects, while Udacity is focused on the sciences, especially computer science. Udacity has an auto-grader while the Coursera course we took [15] only had multiple-choice and short answer questions based on whatever external programming environment the user cared to use. But, two other open educational websites are very similar to ours. The first, Codecademy (www.codecademy.com), launched in October 2011. Initially they only offered JavaScript, but now they also offer Python programming. The second, Khan Academy (www.khanacademy.org/cs), has had online interactive programming lessons since August 2012, although they have offered video-only CS content since 2011 and auto-graded interactive math lessons since 2010.

Both Khan Academy CS and Codecademy are great environments for learning to program. With Codecademy, the student code is retained in the coding window between steps of a multi-part exercise, which gives the user a sense of continuity in progressing with their work. We found that their UI needed improvement in spots, for example there were "hints" that necessarily *had* to be read in order to complete a problem, and occasionally the grader didn't specify the nature of the failed test when a student's code would not pass. They allow community-generated lessons, although we found this made it difficult for a novice to guess a useful sequence of lessons that would teach skills in an irredundant, incremental manner. However, its continuing site development and curriculum refinement appear to be addressing both of these issues.

The design of the Khan Academy CS user interface is drastically different from traditional coding environments, and is inspired by the "responsive programming" ideas of Bret Victor [17]. The ubiquituous Khan Academy video content takes the form of a narrated replay of the lecturer's code window, which can be interrupted at any time and edited. As the user edits, the code is continuously re-compiled and re-executed; numeric constants in the code can be changed by dragging a slider, which creates the effect of an animation when there is graphic output. Any syntax/run-time error comes with automatically-generated suggestions on how it can be fixed. These are fantastic tools but beyond the scope of our own project.

Compared with Codecademy and Khan Academy, our layout is sequential like the traditional concept of a web "page." Each lesson, together with its embedded exercises, is laid out in a vertical sequence on a scrollable web page. Despite its old-fashionedness, it has some advantages over video/audio tracks: it is simpler for authors, can be easier to navigate when reviewing, and it is easier for students to follow at their own speed, including students not fluent in English.

Two other formidable online interactive Python courses date back to 2011. One is pyschools.com, a collection of 210 exercises arranged in 14 topics. While nice, it lacks overall organization and narrative. Another notable site is How to Think Like a Computer Scientist: Interactive Edition [11]. It runs a Python interpreter within the browser's JavaScript interpreter and builds on excellent Creative Commons content from Downey's "Think Python" book [2]. The experience with their site is quite similar to ours. On the one hand not all of their exercises are auto-gradeable, but on the other hand their auto-grader is able to run client-side. The same authors recently released a sequel, in the same interactive format, on algorithms and data structures [12], and published a paper [13] discussing their methodology.

2. BIRD'S-EYE VIEW

The CS Circles website is "end-to-end" in the sense that everything is handled through the browser interface: submitting code, seeing the grader's reply, editing with syntax highlighting, debugging visualization with forward/backward stepping, saving and loading solutions, and asking for help. There is a curriculum to serve as a backbone, but we also make judicious use of flexibility: some lessons are optional, helping each user learn at their own choice of depth; and some lesson groups can be completed in any order, which we think makes the course less monotonic.

Content. The content begins with a lesson on "Hello, World!", including a working click-to-run program and a crashing one. We continue through variables, function calls, comments, quotes, and all of the fundamentals needed in order to write or understand an elementary program. Most lessons have a narrative where a new feature is motivated and explained, with examples, caveats, and exercises for the user. A few special lessons don't talk about language features but instead about practical matters like errors, design, and debugging; several lessons have only exercises based on material from prior lessons, helping to re-consolidate what the user learned previously. The final lessons discuss basic auto-decryption of Caesar ciphers, recursion, object vs. value identity, and efficiency. We forced ourselves not to make the lessons too long; 1000 words was a soft upper limit on what was reasonable to convey in a single sitting. The end goal for our lessons is that the user will have enough knowledge and confidence to understand further information available on the web.

In the top half of Figure 1 we show an example of the first exercise on the site, which is to write a program that prints out a particular message, in analogy with the Hello, world! example. When the user submits their code, it is compiled, executed, and graded server-side. Then, without a page reload, the grading box is updated according to the result of grading. The bottom half of Figure 1 shows the result of a successful submission for this exercise.

There is little "gamification" in CS Circles, since we feel that a public points-and-badges system can be intimidating for some beginners, and can be a distraction in general. But when an exercise is completed, its box turns green and a checkmark appears; the "My Progress" page keeps a hyperlinked visual list of which exercises are completed, so students can easily pick up where they left off or go back over previously-skipped exercises. We receive a lot of positive user feedback; here are portions of unsolicited quotes from two students about their enjoyment of learning Python and solving the exercises.

Jun 11 2012: "*I'm addicted to how the frustration of getting the function or program wrong at first transitions into excitement as I finally figure it out, and it all makes perfect sense laid out in front of me.*"
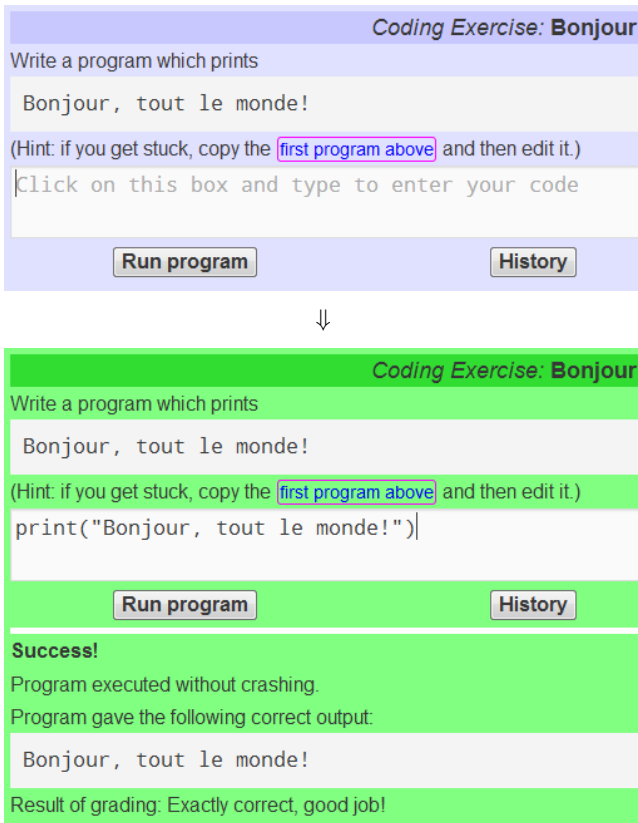
Figure 1: Top: the first exercise in the course. Bottom: when the user submits a correct submission, the grader reply is shown (without a page reload). The box color has changed from blue to green. Not shown is the checkmark appearing in the top right corner.

Aug 14 2012: "*Sites like Codecademy tend to corral you through a safe introduction of 'type this after me' but Computer Science Circles has really helped me apply the things I'm learning in a fun, more thought-intensive way.*"

Our position is that games are addictive because they are fun and engaging, and that copying the fun and engagement is more effective than copying more superficial aspects. Likewise, because we assume that anyone completing our exercises is doing so just for the pursuit of learning and/or problem-solving, we don't take any precautions against plagiarism. This means that in an institutional setting, while CS Circles can be used to deliver content, its exercises should not be used as a way of assigning marks.

2.1 Usage Statistics

In Figure 2 we show three different charts describing usage data for CS Circles. The top chart shows the number of users that have completed each problem on the site, sorted from most completed to least completed. The "outlier" with the least number of completions is the final exercise on the site, which asks the user to look up information from outside of our site and implement the Sieve of Eratosthenes method for computing primes.
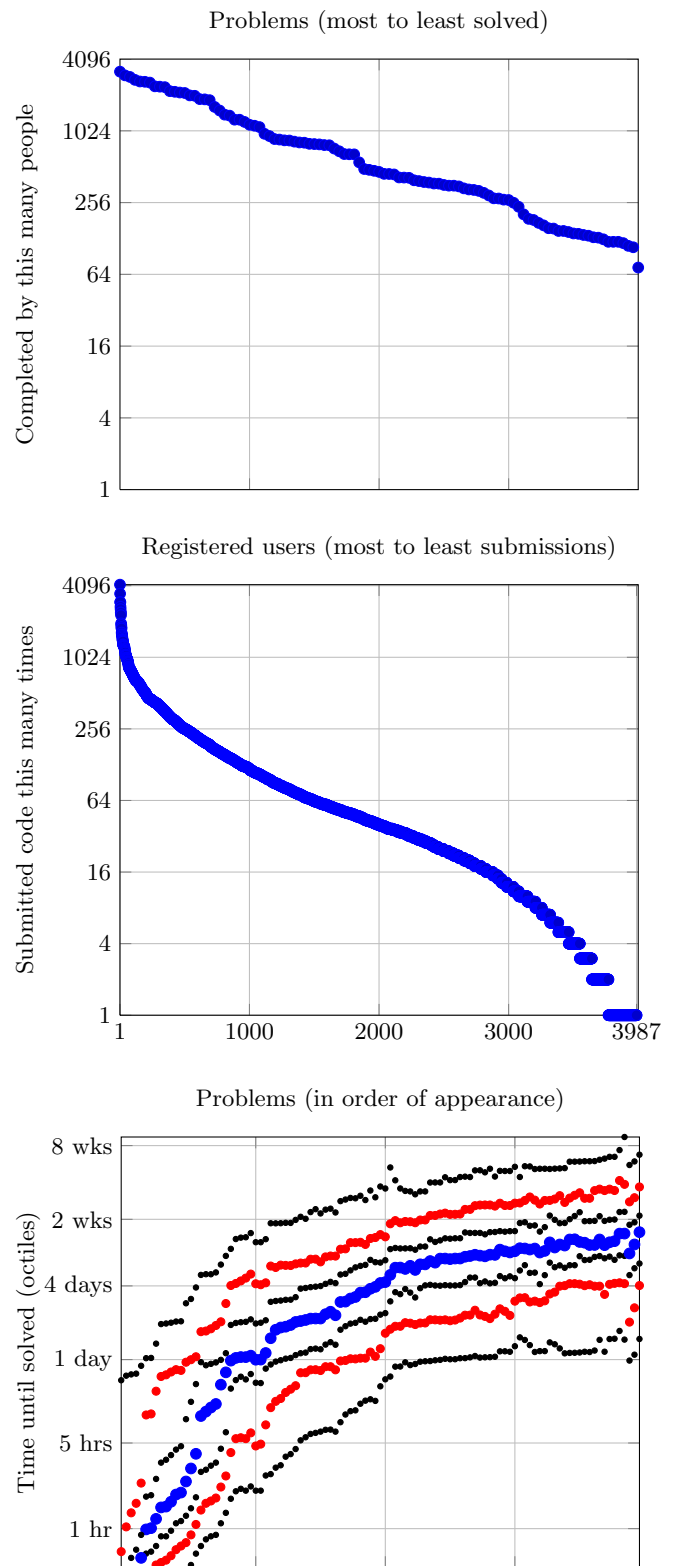

Figure 2: Top: All 105 problems, sorted by the number of people who have completed them. Middle: The users of the site, sorted by the number of times they have submitted code. Bottom: For each problem, conditional on solving it, octiles of time elapsed between registration and completing it.

The middle chart in Figure 2 shows the total number of code fragments submitted by each user. Some traffic is caused by usage of the "console" for non-CS Circles purposes, although an an informal review, this inorganic usage wasn't significant in the top few most prolific submitters.

The bottom chart in Figure 2 shows the typical time elapsed before each problem is completed. Specifically, for each problem and for users who have completed that problem, we show the octiles (median in blue and quartiles in red) of time between account creation and completion of that problem. Note that for *every* exercise, at least half of all completions were achieved within two weeks.

We have no formal biometric data, but in unsolicited feedback and the "Help" system four of our users have self-identified as a PhD student, a lawyer, a parent and child, and an alumnus of U. Waterloo from the 1970s.

3. FEATURES

3.1 Student Tools

Whereas a typical IDE has dozens of buttons our coding windows start with just 3: submit, help, history. The remaining features are introduced later, discussed one at a time in various sections, so that the student knows all of the available tools without reading any manual. These techniques help avoid novices from getting overloaded.

The "Help" button is for the student to use when they are stuck and want targeted assistance. Clicking on this button opens up a new text field where the student is directed to type a few sentences about what they've tried and where they are stuck. This is sent along with the current contents of the code editor for that problem.

For each user, every coding exercise automatically stores a history of all prior submissions. The history can be accessed on demand, and is also available to the student's teacher to help understand what a student has tried, when they ask for help. When a user re-visits a page, the most recently submitted version is automatically loaded in the text editor.

In lesson 4, we introduce the two debugging tools for the course. One is simply a "console" that allows the student to run arbitrary code without any grader interference. Every example has a button which allows it to be copied to the console, in a new browser tab, with a single click. The other debugging tool is an open-source Python "visualizer" [5] that can take an arbitrary program, store the steps of its execution, and replay those steps both forwards and backwards in time along with a visual representation of all of the variables/call stack. This replicates most of the functionality of a client-side debugger but the backwards execution is even more useful than what most IDEs currently offer. Students run the visualizer on 600 pieces of code per day on average.

The ability to specify a test input is introduced in lesson 5, when the command to read from standard input is introduced. In later lessons, when users are tested based on their function definitions, this input box is replaced with a text area for test commands.

In lesson 7, just after introducing the first block-structured elements of Python, we switch the user to a "rich" editor, provided by the CodeMirror project [6]. It highlights syntax, enumerates the lines of code, matches parentheses as the user types, and performs smart indentation. We have customized it to resize automatically up to a fixed limit, but to also be manually sizeable when the user desires.

3.2 Lesson Designer Tools

CS Circles is exercise-centric: indeed, exercises are embedded throughout the lessons. We use traditional exercise formats like short answer and multiple choice as well as coding exercises in many flavours.

The Python auto-grader is similar to existing auto-graders used for olympiads and public problem websites. Aiming for beginners necessitated that the grader offers many more grading styles than the typical stdin/stdout approach. While we teach the `print` function in lesson 0, input reading doesn't occur until lesson 5 and for this reason the early exercises take input by way of variables which the grader pre-defines with values. In lesson 1 the *swap* exercise — write code to put the value of `x` in `y` and vice-versa — has the grader pre-define variables, and test their values after the user code executes. Later, once we have described how to define functions with `def`, inspecting the values returned from function calls becomes the main testing approach. We recommend that each of these three methods (variable-based, stdin/stdout, function-based) has its place in a beginner course.

Our grader uses a number of other techniques in order to maximize the variety and fun of the exercises:

- randomized test cases, with the answer specified implicitly by a model solution, to prevent hard-coded solutions;

- pre-defining non-working initial code, and optionally limiting the (Levenshtein) edit distance to a valid solution;

- pre-defining a function for the student;

- forbidding the student from using some built-in functions;

- requiring that a specific error be produced;

- and plugging-in of custom graders defined in Python (for problems where more than one answer is correct).

Code Scramble. We present a small fraction of our coding exercises as *code scramble* exercises: a solution is already given in the coding window, but with its lines appearing in the wrong order; users are only permitted to drag-and-drop entire lines of code until they find a correct solution. A nice side effect of writing a problem in this way is that we don't rely on the student to have memorized any past syntax, and they can focus purely on the logical aspect of the problem. See also the studies [9, 3] on "syntax-free programming." The problem type we call "code scramble" has been previously called *Parsons problems* [14]; see Denny et al. [1] for a thoughtful quantitative study of how complex these problems should be for optimal effect. Unlike us, they sometimes give decoy lines that must be identified and left out of the solution, or they pair up real lines with decoy lines. Generally, our code scrambles are of medium difficulty: they can be solved most but not all of the way via syntactic analysis. See also [7] for information on *C-doku*, near-complete programs with specific short answer blanks that the student must fill in, with additional ingredients such as code coverage requirements.

Hints are essential to writing an exercise that can meet the needs of the widest variety of students. As a rule, our problems are always solvable using only the tools that have

been previously introduced to the student, even without hints. However, we also want to avoid that students get stuck on a particular problem just because they have overlooked an old idea or they are unable to come up with an innovation on their own. For this reason we offer clickable hints. The most basic version is one whose content appears like a pop-up window when clicked, which can be moved around the page to a convenient place and closed when no longer needed. Occasionally we find that a very conceptually challenging exercise (such as the swap exercise) requires many hints, and in this case we present them in an "accordion" style, where at most one of the hints can be slid open on the page in the midst of the existing content, which avoids overloading the user interface with too many boxes.

Two other features are the result of user feedback: a "cheatsheet" in html and pdf formats, which contains a quick review of all the functions and syntax used by our curriculum; and automatically-generated "previous/next lesson" buttons on the bottom of each page, precluding the need to scroll all the way up to the main navigation menu when a lesson has been completed.

3.3 Teacher Tools

A student can ask a teacher, mentor or friend who has signed up for CS Circles to be their "guru." Conversely, we imagine that for any classroom using our site, the teacher would tell the students to declare him/her as their guru. Currently, there are 8 gurus with five or more students registered under them; in total these gurus have 108 registered students.

A student with a guru can send their "Help" messages to their guru instead of to the CS Circles staff. When a teacher clicks on the reply link sent in the e-mail that they receive, they are brought to the "mail" page of our website. The mail page offers the teacher several tools we found invaluable in providing fast personalized responses:

- the complete submission history for that student on that problem, to see what they've tried that worked or did not, and how close they got;

- a view of the "My progress" page for that student, to see where they are overall;

- a list of the teacher's other messages about the same problem, which allows them to cut-and-paste relevant information in the case several students have the same question around the same time.

We have found that this system is an extremely efficient way to provide targeted, personalized feedback to hundreds of students without much work, filling in the inherent gaps in the help that an auto-judge can provide.

There is another surprising benefit of this "Help" button: as lesson authors, we are able to get a tremendous amount of quick feedback when any newly-introduced content is too difficult or has bad wording. Many of the hints in the current curriculum are the result of boiling down commonly-asked questions received in this way.

4. IMPLEMENTATION

Our website is built on top of the WordPress content management system (CMS). It is estimated[2] to be the most

[2] http://trends.builtwith.com/cms

```
[pyBox repeats=3 precode="people = _rint(10, 100)"
autotests="heads\nshoulders\nknees\ntoes"
solver="heads, shoulders, knees, toes =
people, 2*people, 2*people, 10*people"]
```

$\Downarrow$

Figure 3: Top: a WordPress-style shortcode that defines an exercise in our system. Bottom: the result of a submission sent to the this exercise.

popular CMS, hosting a significant portion of all sites on the internet. We chose it because it has an easy web-based interface for content editing (including instant preview of edits) and management, it has a built-in secure login system we could extend for our own purposes, it is built on a well-maintained and easy-to-modify code base, and it has a plugin system which lets us easily add third-party features. We use plugins for site search, analytics, contact forms, translation, LaTeX, jQuery UI, custom auto-mails, and role management for special users like translators.

As mentioned earlier, CodeMirror [6] and the Python visualizer [5] are two prominent non-WordPress tools that we integrated into our site. Flexigrid[3] is another external library that we use, which allows for smooth and secure user access to database slices — for example this lets you view your past code submissions without pulling them all from our server at once.

Lesson pages are stored, like in most CMSes, as html-formatted text in a database. The auto-grader and exercises are specified using WordPress "'shortcodes." For example, the code at the top of Figure 3 defines an exercise where students must correctly define variables counting the heads, shoulders, knees, and toes at a party, given a pre-defined count of the number of people. The specification, when embedded in the lesson text and rendered, produces the exercise. At the bottom of Figure 3 we show an example of the grader's reply to a partially correct solution. Using this quick text-based specification system has been instrumental in quick prototyping and lesson editing.

The auto-grader and surrounding code is written in a mix of several languages. There is html/js/css to handle browser interaction, PHP to interface with WordPress and gener-

[3] http://flexigrid.info/

ate dynamic html, Python to do introspective grading, and C++ to call the core "safeexec" sandbox routines. The latter was developed starting from the safeexec module of the Mooshak contest system [8]. For a survey of the techniques involved in executing arbitrary user code securely, see the excellent survey of Forišek [4] or the recent new implementation of Mareš and Blackham [10].

The grader has a run-time limit of 1 second by default. With the sole exception of the Sieve of Eratosthenes exercise, we don't intend that any of the coding problems need to be solved in an optimized way. So far the design has been successful: we've not seen any correct student code that times out on any other problem.

We are starting to experiment with using off-site "cloud" computing to reduce the load on our centralized server. It has been very promising so far and in particular our visualization code runs there, falling back to the central server only when the user is not able to communicate directly with the off-site server. We intend to pursue this further, and use caching to help keep the site responsive.

5. FUTURE WORK

Our experiences with WordPress and its great plugin community drive us to ask, would it be useful to distribute the technologies behind CS Circles as a set of plugins? It would be excellent to let teachers and other parties develop and offer online lessons without the need to pay a large company. We would also like to see in-browser auto-graded exercises and sandboxes become part of the on-line documentation for other languages such as JavaScript, R, and Sage.

Interactive input and output is not currently possible in our system, but it would be an important feature to add in a future version. As a way of partially compensating for this, our "Resources" page recommends the `pygame` book [16] and its sequel for students to learn about interactivity.

Currently our website gets traffic from all over the world; about half of it comes from outside of North America, but only 30% of the overall traffic is from non-anglophone countries. We are currently finishing a French translation of the site and are plan to offer it in other languages as well. Programming generally requires enough English competency to make sense of the syntax and the errors, but we are attempting to translate every other part of the site.

Can a useful programming course be taught on a touchscreen/tablet/iPad? Our intuition is that a comfortable keyboard is an important ingredient in making any significant amount of coding enjoyable. However, "code scrambles" might work even better on a touchscreen.

6. ACKNOWLEDGMENTS

7. REFERENCES

[1] P. Denny, A. Luxton-Reilly, and B. Simon. Evaluating a new exam question: Parsons problems. In *Proc. 4th ICER (Int. Workshop Comp. Ed. Research)*, pages 113–124, 2008.

[2] A. B. Downey. *Think Python*. O'Reilly Media, 2012. Earlier version: *How to Think Like a Computer Scientist: Learning with Python*, Green Tea Press, 2008.

[3] C. Fidge and D. Teague. Losing their marbles: syntax-free programming for assessing problem-solving skills. In *Proc. 11th ACE (Australasian Conf. Comp. Ed.)*, pages 75–82, 2009.

[4] M. Forišek. Security of Programming Contest Systems. In V. Dagiene and R. Mittermeir, editors, *Information Technologies at School*, pages 553–563, 2006.

[5] P. Guo. Online Python Tutor, 2011–. http://people.csail.mit.edu/pgbovine/python/.

[6] M. Haverbeke. CodeMirror (Version 2.x), 2011–. http://codemirror.net/.

[7] D. M. Hoffman, M. Lu, and T. Pelton. A web-based generation and delivery system for active code reading. In *Proc. 42nd SIGCSE (ACM Tech. Symp. Comp. Sci. Ed.)*, pages 483–488, 2011.

[8] J. P. Leal and F. Silva. Mooshak: a web-based multi-site programming contest system. *Software: Practice and Experience*, 33(6):567–581, 2003.

[9] M. Lopez, J. Whalley, P. Robbins, and R. Lister. Relationships between reading, tracing and writing skills in introductory programming. In *Proc. 4th ICER (Int. Workshop Comp. Ed. Research)*, pages 101–112, 2008.

[10] M. Mareš and B. Blackham. A new contest sandbox. *Olympiads in Informatics*, 6:100–109, 2012.

[11] B. Miller and D. Ranum. *How to Think Like a Computer Scientist/Learning with Python: Interactive Edition (Using Python 3.x)*. Runestone Interactive, 2011. http://interactivepython.org/courselib/static/thinkcspy/index.html.

[12] B. Miller and D. Ranum. *Problem Solving with Algorithms and Data Structures*. Runestone Interactive, 2012. http://interactivepython.org/courselib/static/pythonds/index.html.

[13] B. N. Miller and D. L. Ranum. Beyond PDF and ePub: toward an interactive textbook. In *Proc. 17th ITiCSE (ACM Conf. Innovation & Tech. in Comp. Sci. Ed.)*, pages 150–155, 2012.

[14] D. Parsons and P. Haden. Parsons' programming puzzles: a fun and effective learning tool for first programming courses. In *Proc. 8th ACE (Australasian Conf. Comp. Ed.)*, ACE '06, pages 157–163, 2006.

[15] T. Roughgarden. "Algorithms: Design and Analysis, Part 1", June–August 2012. *Coursera*. http://www.coursera.org/course/algo.

[16] A. Sweigart. *Invent Your Own Computer Games with Python*. Self-published, 2008. http://inventwithpython.com/.

[17] B. Victor. Inventing on principle. In *4th CUSEC (Canadian University Software Eng. Conf.)*, 2012. (Keynote talk) http://vimeo.com/36579366.