MOOCs on Introductory Programming: A Travelogue

Mordechai (Moti) Ben-Ari

The past few years have seen several initiatives to implement *Massive Open Online Courses (MOOCs)*. I decided to participate in two MOOCs as a student while simultaneously evaluating the courses as a CS educator. My goal was to assess the educational experience provided by MOOCs: the effect of this online technology on the learning process and the quality of the pedagogical methods used by the instructors. This paper is a travelogue relating my experiences and the conclusions I draw from them.

1 THE COURSES I STUDIED

MOOCs became widely known through the course on artificial intelligence given by Sebastian Thrun at Stanford University. This led him to found a company, Udacity [10], to offer MOOCs. In the summer of 2012, I took the course *CS101: Introduction to Computer Science: Building a Search Engine* given by David Evans of the University of Virginia. Subsequently, in the fall 2012 I participated in a similar introductory course from Coursera [4], *An Introduction to Interactive Programming in Python* by Joe Warren, Scott Rixner, John Greiner and Stephen Wong of Rice University.

Both courses teach introductory programming using Python. The fact that I did not know Python contributed to the authenticity of the learning experience. I watched the video lectures, answered the quizzes, and submitted all the programming exercises and projects.[1]

[1] *Important Note:* Since both organizations offer numerous courses, my comments should not be taken as referring to the organizations themselves nor to any other courses that they offer. In this paper, I use "Udacity" and "Coursera" as a shorthand for the specific courses I studied in 2012. Although I criticize many of the design decisions, it is clear to me that the decisions were not arbitrary and that the developers of the courses took these decisions for logically reasons. The aim is to present my evaluation and to show how a student might feel about these decisions.

2 OVERVIEW OF THE COURSES

The Udacity course consists of video lectures by Evans, multiple-choice questions interspersed within the lectures, and numerous programming exercises. The context of the lectures and many of the exercises is building a search engine. Programming is done within a browser window and the programs are saved in the cloud. Assessment is performed automatically.

The Coursera course consists of video lectures, primarily by Warren and Rixner, with a few lectures given by Greiner. After each group of three lectures, there is a multiple-choice quiz and after two such groups there is a mini-project. The context of the *mini-projects* is interactive games. Programming is done within a browser and programs are saved in the cloud. Assessment of the quizzes is automatic, while peer assessment is used for the mini-projects.

3 SCHEDULE

The Udacity course consists of video lectures by Evans, multiple-choice questions interspersed within the lectures, and numerous programming exercises. The context of the lectures and many of the exercises is building a search engine. Programming is done within a browser window and the programs are saved in the cloud. Assessment is performed automatically.

4 LECTURES

The lectures are excellent and I sensed the enthusiasm and commitment of the lecturers.

Udacity has a large number of very short 2-5 minute lectures, while the Coursera lectures are longer, six 10-15 minute lectures per unit. The lectures are streamed but the Coursera lectures can also be downloaded, which is important for students without a reliable internet connection. (After I completed the course, Udacity enabled downloading their videos.)

The technology of the lectures did not go beyond an electronic whiteboard and the projection of the display of a computer. Most of the material is hand-written, while I would have expected slides to be prepared beforehand and available for download. More than once I had to search within the videos for some detail concerning the programming projects and this is much more difficult than scanning printed material or searching electronic documents.

5 PROGRAMMING ENVIRONMENT

In both courses programming is done within the browser. This is becoming quite popular (see, for example, the upcoming version of Scratch) because it avoids the glitches of installing software.

Udacity's system was quite slow, so eventually I downloaded the interpreter for Python 2.7 and installed it locally. Fortunately, Coursera's system, CodeSkulptor, was very responsive, since they did not enable downloading of an interpreter for their modified dialect of Python 2.6 with their own GUI API. The Coursera environment conveniently contains links to basic documentation for Python and the API.

Coursera does not maintain a mapping from students to their programs stored in the cloud. You are instructed to bookmark the urls and/or to save them (or to download the source code itself). I am sufficiently experienced to backup frequently, but I am sure that some novices must have suffered the loss of their work.

6 DEBUGGING

There are no facilities for debugging nor is the subject taught in these courses! You cannot trace the execution nor set breakpoints. The lectures do not mention techniques such as commenting out parts of a program when searching for a bug or creating a short program to check a suspicious construct. They do not even suggest the simple technique of scattering print statements throughout a program. The Coursera course contains a few lectures demonstrating common bugs and how to fix them, but there is no instruction on how to diagnose bugs.

7 PROGRAMMING CONTEXT

Both Udacity and Coursera attempt to increase students' motivation by basing the courses on interesting topics.

The Udacity course was based upon building a search engine. I did not find the topic itself particularly motivating, because the programs require primarily bread-and-butter string and list processing. The search engine is built incrementally so that the student need only add a few lines to an existing program. To avoid accessing the internet every time the program is run, a test page is given as a single very long string; unfortunately, this makes debugging difficult.

The Coursera course asked the student to develop games such as Pong, Blackjack and Asteroids. The instructors admit to being veteran gamers, but not everyone will find games motivating. The advantage of games is that the graphics and interaction are exciting; the disadvantage is that the student must master GUI programming and such programs tend to be rather long. The mini-projects each required 100-200 lines of code, which is not really "mini" for novices. However, the projects are well structured as small incremental tasks that a diligent student can complete.

In both courses, there is a lot of scaffolding so that the student is not challenged to design a program from scratch. Udacity does include a very large number of small (and optional not-so-small) programming problems so this is less of a problem. I believe that small programming exercises are important for solidifying students' grasp of programming constructs, so I find Coursera's restriction to programming projects problematic.

8 CONTENT

The content of the two courses is almost entirely programming. Evans does talk about complexity issues concerning the internet, and includes lectures and interviews on historical aspects of computer science. Neither course discussed program design that is central to learning software development. (There is one minor exception: a very short discussion by Warren and Rixner on the design of the Blackjack program.)

The Coursera course can be characterized as objects-late, as objects appear about half-way during the course, although I question the usefulness of teaching objects in the absence of instruction on program design. The Udacity course does not include object-oriented programming.

9 FLEXIBILITY

Any instructor knows that a student's question can change one's teaching by uncovering a misconception you hadn't thought of before or a misunderstanding of one of your examples or explanations. Therefore, if a course is not to stagnate, it must be flexible. Furthermore, one has to take into account that solutions will eventually be posted on the internet and available to future students.

Unfortunately, MOOCs are quite inflexible because it is difficult to redo video lectures and to change online assignments and examples. Nevertheless, one can distinguish a significant difference between the two courses. The Udacity lectures are very short so additional ones can be added easily and existing ones can be redone, while the Coursera lectures are longer and thus harder to change. More importantly, many of the Coursera lectures concern the mini-projects: Entire lectures are devoted to the rules of blackjack and to design of the asteroids game. Modifying the Coursera course would thus require an almost complete re-implementation.

10 STUDENT ASSESSMENT

Assessment is a major obstacle to the success of MOOCs that are truly massive. This is especially true if the institution offers certification for students who complete the course. Udacity does offer a nice certificate though it remains to be seen whether this has any value. Coursera (Rice University, in this case) offers no certificate.

Udacity's assessment is fully automatic which is easy to do for a course in programming. Of course, this means that only the correctness of a program is assessed and not its style.

Coursera opts for peer assessment where each student is expected to assess the projects of five other students. The advantage of non-automatic assessment is that more can be checked than with automatic assessment, so you can imagine my shock when I found the following sentence in the instructions for peer assessments:

> You are not grading people's coding style. In fact, most of the time, you should not need to look at the code at all in order to do the peer assessment.

I understand their fears – that beginning programmers are not competent to assess style – but that instruction negates any advantage that peer assessment has over automatic assessment.

So what does peer assessment consist of? My favorite example is:

> 1 pt - Program prints blank lines between games.

While the above example is extreme, almost all grading rubrics were similar and could be done automatically, although the implementation would not be easy for interactive programs.

There did not seem to be any procedure for appealing an assessment. On one project most assessors lowered my grade because my program wasn't "always running" as required; apparently, my interpretation of that requirement was different from theirs. If the grade had been important to me, I would want to be able to appeal it.

11 PEDAGOGY

My biggest disappointment came from the complete absence in both courses of any pedagogical innovation. There is no evidence that the instructors are familiar with educational research (for example, on misconceptions), not even at the level of pioneers of CS education research from 25 years ago (cf. the review in [9]). Furthermore, the instructors use no modern CS-specific educational technology, such as pedagogical development environments like BlueJ [2] and jGrasp [6] or program visualization systems like Jeliot [7] and UUhistle [11] that have been shown to improve learning [1]. In effect, I see no pedagogical difference between these courses and the programming course I taught as a teaching assistant over 30 years ago.

12 LEARNING MATERIALS

(Disclaimer: I have authored several textbooks.) Neither course recommended a textbook; in effect, the only source of information was the video lectures and, in the case of Coursera, the linked reference documentation. I see this as a very serious problem: There are times when a lecture glosses over details and there are learners who prefer reading over listening to lectures. I believe that lectures should be supplemented with reading, because it is easier to read and re-read difficult passages until you understand them.

A related issue is that of persistence. I still have my freshman calculus textbook and can use it to look up a forgotten formula, but I doubt that the lectures, source code files and forums of these courses will last as long.

13 SUPPORT

Many years ago I obtained a certificate in business administration from our Open University. It involved academic courses in economics, marketing, finance and so on. All these I studied at home, but attendance was required at a seminar where we were asked to write a business plan as a capstone project. During the seminar I asked a question and the instructor immediately shot back: "You never went to any tutorial sessions, did you?" The answer to my question was simple and the fact that I didn't know it was not due to the absence of any content knowledge, but because I hadn't learned to *think* as an economist. Presumably, I would have learned to do so during interaction with an instructor.

The ability of students to learn to think like computer scientists and software developers depends on adequate interaction with instructors. Both courses depend on forums, which I have never liked: they are too active to be worthwhile reading routinely, and searching is often inadequate to locate the pearls from the dross. The only time I asked a question (pertaining to the semantics of Python list operations), I didn't receive a correct answer; instead, I had to consult the Python language reference to find the answer, a solution that may not be appropriate for novices.

14 SUMMARY

The Coursera course is very demanding, asking students to develop relatively long interactive game programs according to a strict schedule. The development environment was very good, but there should be an option to work offline in case a student lacks adequate and continuous internet access. The peer assessment I find incomprehensible. My impression is that the course would be appropriate for a highly-qualified student who can dedicate the time to work according to a strict schedule, in other words, a student who could attend a university as a traditional student. It is also appropriate for people with professional education and experience looking to expand their knowledge [5]. My experience was consistent with what was reported in that blog: almost all the programs I assessed were well written and fully functioning with at most minor bugs; I find it hard to believe that they were written by true novices.

The poor performance of the Udacity environment needs to be solved; however, since the course uses a standard Python dialect, it is not hard to work offline. The advantages of Udacity are the flexible time schedule and the plethora of automatically assessed programming exercises. In my opinion, a non-traditional student could achieve a good beginner's-level knowledge of Python programming from this course.

15 WHAT OTHERS SAY

Surprisingly, very little has been written on MOOCs and most references I found were blogs and editorials. These references are consistent with what I found from my experiences: "In fact, the absence of serious pedagogy in MOOCs is rather striking, their essential feature being short, unsophisticated video chunks, interleaved with online quizzes, and accompanied by social networking" [12, p. 5]. More significant is Butin's comment on the labor-intensive process of teaching:

> Much like an orchestra today still takes the same amount of time to play a Mozart symphony as it did in 1800, there is a seeming productivity cap in higher education due to the labor-intensive process of actually teaching students something. Technology seemingly cannot simply transform or supplant or speed up the deeply dialogical process of student-teacher give-and-take [3].

The most interesting paper was by Fred Martin. He participated in Thrun's artificial intelligence course together with 16 of his students at the University of Massachusetts, Lowell. Off-loading the lectures and the mechanics of exercises to the MOOC significantly improved his teaching:

> Instead, we used in-class time for conversations about material that people found confusing or disagreed upon. We had some great discussions over the course of the semester [8, p. 27].

Martin calls this the *flipped classroom*, quoting Daphne Koller, a founder of Coursera.

I encountered an unintentional flipped classroom as a student many years ago, when I studied a course in differential equations. The university assigned a professor who was world-famous in his narrow specialty, but who admitted that he hadn't studied differential equations since college. He proposed that we (professor and students, alike) read the textbook and then use the class time to work out problems together. That course was a far better educational experience than the massive introductory physics course with hundreds of students in a lecture and graduate-student assistants chosen for their potential to become research scientists and not their talent as educators.

16 CONCLUSIONS

The online technology proved to be constraining rather than enabling. The absence of debuggers and textbooks I find extremely problematic, and neither of the two options for assessment – automatic assessment of limited scope or superficial peer assessment – are truly effective.

The ideal use of MOOCs seems to be in a flipped classroom, which is not so different from the classical distance education of open universities. In an open university, students read textbooks and listen to video lectures; however, their homework is graded and they receive feedback from teaching assistants who are also available to answer questions, either remotely or at (optional) tutorial sessions. The problem with environments like flipped classrooms is that they lose the convenience, financial savings and broad accessibility that MOOCs are supposed to provide.

Finally, if MOOCs are to fulfill their promise of improving education, they must adopt and adapt pedagogical innovations and educational technology. **Ir**

References

[1] Ben-Ari, M. et al. "A decade of research and development on program animation: The Jeliot experience," *Journal of Visual Languages and Computing*, 22,5 (2011): 375-384.
[2] BlueJ. http://www.bluej.org/.
[3] Butin, D.W. "What MIT Should Have Done," *eLearn* 6 (2012).
[4] Coursera. https://www.coursera.org/.
[5] Guzdial, M. "Who completes a MOOC?" *Computing Education Blog*, 25 September 2012. http://computinged.wordpress.com/2012/09/25/who-completes-a-mooc/ Accessed 28 March 2013.
[6] jGrasp. http://jgrasp.org/.
[7] Jeliot. http://www.cs.joensuu.fi/jeliot/.
[8] Martin, F.G. "Will massive open online courses change how we teach?" *Commun. ACM* 55, 8 (2012): 26-28.
[9] Robins, A. et al. "Learning and Teaching Programming: A Review and Discussion," *Computer Science Education*, 13, 2 (2003): 137–172.
[10] Udacity. https://www.udacity.com/.
[11] Uhistle. http://www.uuhistle.org/.
[12] Vardi, M.Y. "Will MOOCs destroy academia?" *Commun. ACM* 55, 11 (2012): 5-5.

MORDECHAI (MOTI) BEN-ARI
Department of Science Teaching
Weizmann Institute of Science
Rehovot 76100 Israel

Categories and Subject Descriptors: K.3.1 Computer Uses in Education: Distance learning; K.3.2 Computer and Information Science Education: Computer science education
General terms: Human factors
Keywords: Massive open online course, MOOC, Python

DOI: 10.1145/2465085.2465102

A THIRD COURSE

During the winter of 2013, I participated in the Coursera course *Control of Mobile Robots* by Magnus Egerstedt of the Georgia Institute of Technology. The enthusiasm of the lecturer was apparent and he prepared slides that could be downloaded. I truly missed having a textbook as it was difficult to study the mathematical details from the slides and lectures. Students were encouraged to solve programming exercises using Matlab, but these were optional. Although a student discount was arranged, I assume that the "openness" of the course meant that purchasing the software could not be required. Furthermore, no instruction was given in Matlab itself. Overall, the experience was similar to that of the other courses: high-quality learning materials that are appropriate for the fully independent student or for students in a "flipped classroom," but less appropriate for a student who needs textbooks, more exercises and direct interaction with an instructor.