Teaching Statement – Cyrus Omar

When I teach, my goal is to help each individual student develop an understanding of, and appreciation for, fundamental principles. By this, I mean fundamental mathematical and scientific principles, of course, but also fundamental principles of technical discourse, because I believe that universities must deliberately aim to raise the standard of discourse in the broader culture of computing.

For example, I believe that being able to use a shared vocabulary precisely and fluently is of fundamental importance. When a student in a recitation section or in office hours has a question about a function or proof that they are working on, I ask them to read and explain their code out loud to me "from the top" and, as they do so, I help them rephrase their reading until it is clear and correct. This prompts them to organize their thoughts, so that by the time they have finished, many students have figured out the next step on their own. If not, they are better prepared to understand my subsequent guidance. The benefits of this approach extend beyond the classroom, because the very same communication skills help students engage productively with their future coworkers and collaborators.

I strive at the same time to communicate with *patience*, *sensitivity* and *generosity*, and I believe that we must explicitly advocate for these virtues, just as we explicitly advocate for precision and rigor, if we are to build a computing culture that does not alienate those who are new or struggling with subtle concepts and unfamiliar standards of discourse. Often, these are individuals from underrepresented groups who have not previously had access to a supportive technical community. I believe that great universities have a cultural responsibility to lead in this regard.

In order to demonstrate that the principles that students are learning are in fact quite practical, I like to tell vivid stories, including success stories from industry, stories about cool research projects that build upon what the students are learning, and stories that demonstrate the perils of taking an *ad hoc* approach to software development, language design and technical discourse. I am also a big proponent of course projects, and of undergraduate research more generally. I have mentored three undergraduate summer research students, and all of them were able to publish their results and went on to prestigious PhD programs in programming languages (at CMU, Washington and Northeastern.) I also helped advise a student working on a senior thesis who recently started an industry position at Facebook.

Experience

I have served as the head TA for two courses at CMU. In both cases, these courses were entering a period of significant transition, and I was able to substantially influence their development. For my role in these two courses, I was awarded the School of Computer Science's **Alan J. Perlis Graduate Student Teaching Award** in 2013.

Functional Programming (15-150) I served as the head graduate TA, alongside 14 undergraduate course assistants, for the first full-scale instance of CMU's new core sequence course on the theory and practice of functional programming. We all held office hours and weekly recitations, graded code (in Standard ML) and handwritten proofs, and contributed to the development of the homework assignments. The students in this course responded quite positively to my teaching style — I frequently needed to reserve a separate classroom for my "office" hours, because 20 or more students (out of about 180) would invariably attend. In addition to these responsibilities, I served as a guide to the undergraduate course assistants and participated in the preparation of grading rubrics, recitation notes and exam questions.

Principles of Programming Languages (15-312) I served as the senior graduate TA, together with another graduate TA, for CMU's undergraduate programming languages course. This course, taken by about 45 juniors and seniors, was taught by Prof. Bob Harper, who had just released his book *Practical Foundations for Programming Languages*. As such, we substantially revamped every homework assignment to follow the developments introduced in this book (which is now in its second edition and being used at an increasing number of universities.) I played a leading role in this transition, developing most of the theoretical problem statements and reference solutions and contributing significantly to the programming assignment code. I also held office hours, lectured in weekly recitations, developed new material for these recitations and responded to questions on the web forum. I also had the opportunity to lead one lecture, where I introduced the fundamental concept of substitution and led students through the proof of the Substitution Lemma.

The feedback from my students was overwhelmingly positive, for example:

> "Very patient when one on one, good explanations, helpful. Stays even after office hours technically over to answer questions, very good TA"

> "Know's what he's talking about and knows how to explain it."

Future Courses

The experiences described above have left me well prepared to teach core courses covering Functional Programming and Programming Languages (at both the undergraduate and graduate level.) I am also confident that I can teach the necessary prerequisities (i.e. discrete mathematics), and core courses in imperative programming (I have followed the development of CMU's companion Imperative Programming course, 15-122.)

Other undergraduate courses that I believe that I can teach include:

- Programming for Scientists (and Data Scientists). Here my research experiences and coursework in neuroscience would likely be of particular relevance.
- Parallel Programming and High-Performance Computing. I have substantial coursework in this area, as well as substantial practical experience and research experience with CUDA and OpenCL, and non-trivial experience with various other frameworks (e.g. Spark, Hadoop and others.)
- Compilers
- Logics & Proofs, covering various logics, their accompanying proof theories, and proof assistants (e.g. based on Pierce's *Software Foundations*.)
- Introductory Statistics and Machine Learning. I have substantial experience with these topics, both as a scientist and with my more recent research on statistical models of programs and edit actions.

At the graduate level, I can teach core courses on programming languages, type theory, logic and proof theory. I would also love to lead graduate seminars covering the history of adaptable and intelligent programming systems (cf. the definitions in my research statement), and a survey course covering programming language design throughout the history of computing. I would also be interested in contributing to various methods courses for graduate students.

I am also quite excited about the possibilities of incorporating the artifacts of my ongoing and future research into the classroom. In particular, I am working on two artifacts of note:

1. `typy`, a statically typed language embedded into Python as a library. This language was described in our paper at GPCE 2016. This could potentially resolve a perennial tension: should we teach using a language that students are likely to see in industry (e.g. Python) or a language with a cleaner and richer semantics. With `typy`, you get both.
2. Hazel, the artifact that will serve to organize much of my future research. It should be possible to teach an introductory functional programming course using Hazel in the near future. Slightly further out, it should be possible to teach a programming languages course and a programming course for scientists using Hazel as well.

In both cases, I believe that using these artifacts for teaching will not only expose students to exciting new tools, but feed back to help my research group make progress on their design.