Teaching Statement
Stefan Muller

Teaching has been a fundamental and formative part of my undergraduate and graduate careers, and I intend for it to be a fundamental part of my future career as well. In my teaching, I strive to emphasize elegant ideas and theory while also using applications and demonstrations so students can see the impact of what they’re learning from day one.

Teaching Experience

I have had taught nine classes, one as an instructor of record and eight as a teaching assistant. I was the instructor of record for Carnegie Mellon’s Principles of Functional Programming in the summer of 2018, a semester-long course taught over six weeks, during which I lectured almost every day, developed exams and supervised six undergraduate TAs in assignment design and grading. Prior to that, I was a teaching assistant for three semesters at Carnegie Mellon and five semesters at Harvard, during which my responsibilities variously included teaching recitation sections, leading office hours, grading and some course development.

Teaching Objectives

Applicability. One of my goals as an instructor is for students, even in more theoretical courses, to apply the course material to real-world problems as well as their own interests. In my past teaching, I have achieved this objective in two ways: by introducing course materials that allow students to apply the content to various other domains, and by using real-world examples in my teaching. For a functional programming course at Harvard, I developed an assignment in which students build a small computer music library using “lazy streams” in OCaml. In addition to furthering the learning objectives of the course (“lazy evaluation”, differing data representations), the assignment also gave students the feeling that they were doing something real: the stream representation is similar to how music is generated in applications such as video games. Furthermore, the payoff is clear and satisfying: at the end of the assignment, the code outputs a MIDI file and allows the students to hear the music they have generated.

Students enjoyed the assignment at Harvard (where it was still in use for several years after I left), and so I reused it at Carnegie Mellon during my summer teaching, with an additional twist: while the students were completing the assignment, I did an in-class demonstration of a program (which I developed as a case study for my research) that uses lazy streams to represent audio data received over a network and pass it to a sound library. The students enjoyed seeing how programming techniques they learned in class could be used to write real software they might use in day-to-day life, in this case a streaming music client!

In my future teaching, I hope to continue introducing course materials and demos that apply the course content to real-world problems as well as domains like art and music. Depending on the class, I might also introduce a project component in which the students are invited to apply course content to a problem or domain of their choice. In addition to increasing the students’ enthusiasm for the course content, doing this will also make the content more approachable from students with backgrounds in different fields.

Big ideas over small details. When leaving my classes, I also want students to come away with an understanding of an important idea in computer science and how it relates to the course material, to their other classes and to areas of their future work and/or research. For example, one big idea I tried to convey in the functional programming class at Carnegie Mellon was the importance of abstraction, which I taught using an example not even drawn from computer science: every car is different under the hood, but someone
who has learned how to drive one car can easily transfer this knowledge to another car because they have the same interface (furthermore, many people know how to drive cars but few know how to build them). I like to design assignments that emphasize these ideas (e.g., an assignment emphasizing abstraction might have students program many implementations of a given interface). In designing exams, I try to focus on testing understanding of these larger ideas using questions that get at common misunderstandings of them, without relying on minute details of, say, the syntax of a programming language. To make this possible, I believe in open-book exams and/or allowing students to bring sheets of notes, to minimize the amount of memorization and focus their studying on implementations of larger course goals.

**Problem solving.** Finally, I would like for students to develop or hone problem-solving skills in any course I teach (particularly for graduate classes where students are training to become independent researchers, or upper-level undergraduate classes where students might be considering graduate school). This may mean preparing students to design algorithms to solve open-ended problems, or to prove properties of small snippets of code, or other types of problems depending on the course. Students develop this skill largely through practicing it on problem sets and exams. For example, in the functional programming course at Carnegie Mellon, most assignments come with fairly open-ended programming tasks as well as several proofs. Developing the skill of problem solving is not easy. As an instructor, I have encouraged students to collaborate on open-ended problems (within an appropriate collaboration policy) when they get stuck, and to attend office hours. As a TA leading office hours, I spent extended periods of time sitting with students, asking questions to spur their thought process toward a solution without giving one away. The value of this to students has been reflected in my consistently good teaching reviews, even from students who have only attended my office hours. Because resources such as peers and TAs are not available on exams, when developing exams as an instructor and a senior TA, I allowed students a choice of several open-ended problems or proofs so it would not hurt them to get stuck on one.

**Future Teaching**

I am excited to teach more classes as a faculty member, and my experience and interests are compatible with a wide range of teaching assignments. I find teaching introductory classes particularly rewarding, including both more theoretical classes like the ones I’ve taught as well as classes more focused on systems or practice (in teaching more systems-focused courses, my goal of “combining theory and practice” would mean connecting the practical aspects of the course to underlying theoretical principles). I also have the experience to teach many other courses in programming languages and algorithms. For example, I would be interested in teaching an undergraduate-level algorithms course, especially one that (like a class for which I was a teaching assistant at CMU) teaches teaching algorithms and data structures as “parallel by default”, in order to expose students to the increasingly important topic of parallel computing early in their careers. I can also teach upper-level undergraduate and graduate courses in programming languages, including more specialized seminars on topics such as language-based security, which has been an interest of mine since my undergraduate thesis.

At some point, I would also like to teach an upper-level undergraduate or graduate class on practical functional parallel programming, leveraging ideas developed in my research to show students how to develop real-world software that is elegant and maintainable. This last course would be a good opportunity to assign a project in which students apply the course ideas to domains and problems of their own choice, or related to their own research (especially in areas like systems and AI). These projects could easily lead to new and exciting research areas, and I believe would help motivate more students to learn important theoretical foundations as well as advance the state of the art in programming language techniques.