Teaching Statement

Kristopher Micinski, Syracuse University

Passion for teaching motivates every aspect of my scholarship. Since completing my PhD, I have taught 12 courses at the undergraduate and PhD level, consistently achieving top teaching reviews and producing freely-available content (on YouTube) which has gained significant popularity (thousands of views) outside of Syracuse. Central to my approach is a project-focused style, which pairs hands-on programming exercises with challenging projects that demand students to cultivate skill in debugging and software engineering. I consistently innovate in course delivery, leveraging my computing expertise to build highly-interactive courses. However, I also realize that effective teaching requires humility, compassion, and communication. I check in with students frequently and personally, pushing them to achieve as much as I know they can. And I reflect on the ways in which I fall short, learning from my failures and helping me become better at these things.

Teaching History and Goals I have taught a broad array of courses including introductory computing (in Python), programming languages (at Maryland, Haverford, and Syracuse), compilers, and graduate seminars in program analysis and security. After my PhD, I further committed myself to teaching by accepting a teaching-focused postdoc (visiting professorship) at Haverford College. In each of these experiences, I have observed a unifying challenge: undergraduate students have weak end-to-end problem-solving skills, reifying as an inability to successfully identify the proximate source of a bug. This has inspired the following overarching goal: *equip students with the the problem-solving (i.e., debugging) skills they need to write correct, robust programs on their own.* To reach this goal, I design engaging programming projects which tackle foundational CS concepts and build upon collaborative in-class exercises. For example, during my module on the λ-calculus, students build a Church encoder, compiling a significant subset of Scheme into just the λ-calculus; students say this allows them to viscerally experience the Turing-completeness (of the λ-calculus) in a familiar language. While I design these projects to reinforce course topics, an unstated goal is to force students to grow practical debugging skills.

Flipped-Classroom and Course Delivery Early in my career, I focused on developing thoughtful lectures, hoping to inspire students in the same way I was inspired to love computing. As I gained experience, I came to understand the myriad learning styles my students have. However, I observed one common issue: *once a student gets lost, they will quickly feel defeated and become disinterested.* I structure my courses to directly confront this problem in several ways. First, I developed my undergraduate course as a series of 15–30 minute video lectures. While this required serious time investment in filming, editing, and production, students universally report that engaging, well-produced videos encourage them to repeatedly absorb the most challenging concepts. I am proud of these videos, and (if you are so inclined) I invite you to watch one (e.g., "L5: Recursion over Lists") to get a sense of my teaching style. During class proper, I begin by recapping the video, giving a slightly different perspective. However, I also integrate in-class worksheets and programming exercises. These exercises have significantly helped my course outcomes: instead of students becoming cumulatively more lost through lecture, exercises force students to "check in," and force me to devote more time to topics that students find most confusing.

Building a Growth-Positive Classroom Effective instruction requires motivating students to push themselves beyond their limits. I want my students to confront failure as rapidly as possible, understanding these failures as a necessary vehicle for growth rather than a rebuke of their efforts or person. In the beginning of my career, I often wasted my time perfecting content that was useful only to the highest-achieving students, and failing to help lower-achieving students meet their potential. There is no technical solution this problem, because it is a human problem. As such, I *use technology to automate repetitive teaching tasks to enable an unprecedented level of personal attention to students*. I am always available in office hours and email. I directly email students who are lagging on projects, asking them when they plan to finish and emphasizing that I want them to succeed. My teaching reviews reflect this: students *often* say that I am the most supportive instructor they have ever had, and praise my commitment to fairness and student support.