Teaching Statement

Austin Cory Bart
https://acbart.com

During my first semester as an undergraduate Teaching Assistant, I clearly recall a night where I tutored three pairs of students back-to-back-to-back until 2am. It was one of the most satisfying experiences of my undergrad. As an instructor, I can still feel that excitement as I watch students grapple with issues, try out new ideas, and tame unruly code. I consider my teaching to be my most important work, and the best tool I have for improving the world. Even my research has the same goal of systematically improving education and supporting learners.

Computing gives its users new powers. It is literally the closest thing we have to magic. Its practitioners can accomplish things they could never do before, even at an introductory level: build interactive websites and games, find answers to questions at scale, and to solve real-world problems. I believe that learning to think Computationally changes the way that people approach the world, and is a necessary piece of a broad liberal education. Students need to learn how to solve problems, seek help, and learn beyond what they are taught. There is a beauty and majesty to computing, and it is my job to share those feelings with students. When they progress further into computing, they can accomplish even more amazing things. Develop software that changes the world! Build systems so complex that it is impossible to understand them all at once! Write proofs that are fundamental to the very way we compute things! Everyone can benefit from learning these things, if we can only find the ways to reach students.

In graduate school, I took a four course certification in the Learning Sciences. These classes on academic motivation, instructional design, and theories of learning were completely new territory for me; in some ways, the knowledge I gained there has been more valuable than anything I have learned in a Computer Science classroom. It taught me to approach curriculum development with the same kind of strategies that I develop my software with, using data, theory, and practice to iteratively improve my teaching. I am constantly reading literature and research to improve my methods, and regularly practice my skills and obtain feedback.

I believe that lecture is a small part of a much bigger learning process, so every lesson I develop incorporates a blend of presentation, participation, assessment, and feedback. Until students are actively participating in constructing their understanding, they have not had a chance to learn. It might be on paper, it might be out loud, or it might be in a code editor - the important thing is that students need to externalize their understanding and receive feedback. Collaboration, group work, and independent learning are useful tools for facilitating this interaction cycle at scale. Automatic feedback tools, which I have developed in my research, are also crucial for managing my larger courses, but there is often no substitute for sitting down next to a student and socratically walking them through their problems. I plan every classroom experience to maximize the time I spend interacting with students - not just speaking at them, but engaging in a dialog, both one-on-one and in small groups.

When designing lessons, I find multiple ways to motivate students: beyond making the material interesting and presenting with vigor, I draw the big picture of how this can benefit their short-term understanding and long-term career needs. For example, in the Introduction to Computational Thinking course, I incorporate multiple educational contexts: fun computational puzzle games (e.g., where you must guide an avatar to escape a maze) followed by serious data science questions. I always try to balance the challenges of the lesson so that students can have the perfect level of struggle and feel that they are solving worthwhile problems. I let them know that I am always available to them to guide them to success, whether in the classroom, via email, or in office hours. But most of all, I encourage them to stake a claim in their learning and find opportunities to incorporate their own passions. For example, in the final project of the course, students tackle a massive research question of their own choosing, using a dataset they select from their own discipline. Although they all use the same general computational methods, it is up to the student to devise their own questions, build up their visualizations, and produce an answer. Every student has a rich background and potential to draw on, and I find it is valuable to differentiate the lessons to their benefit wherever I can.

Assessments, whether formative or summative, are not an afterthought, but the bedrock of my teaching. Whether my software or my students, testing is necessary to finding problems. After I have written my learning objectives, but before I ever make a PowerPoint slide or a hand-out, I am thinking about what questions will elicit their understanding. Developing an environment where students feel comfortable demonstrating their current abilities is crucial. I combine authentic assessments (e.g., large scale coding projects) with short quizzes that assess their verbal knowledge and intellectual skills. I also use pretests and posttests, not just to measure my students and show them their growth, but to measure the validity of my methods.

I expect a lot out of my students, and I feel that they should expect a lot out of me I need to give them struggles that they are capable of overcoming. I need to be flexible during lessons and the semester to adapt to my students' abilities and desires, and even shortcomings in my instruction. I need to be fair to each of them, and assign an appropriate amount of work. I have no problem when my students attempt and fail - it is my job to help them pick themselves up and try again. I believe mastery learning is essential to helping students develop their skills. I don't care about grade point averages and late point deductions, I care about whether my students walk out of the door with an improved understanding of the world.

Ultimately, the instructor creates the best environment that they can for their learners, but this is all they can do. There are no optimal algorithms to the classroom, and sometimes you have to lose in both time and space. Still, I think there are always tremendous opportunities to inspire, develop, and nurture students, if we're open to them. Computer Science is a discipline with infinite potential, and we are only just beginning to figure out how people learn its many topics. I will not pretend that I have the answers, but I am always excited to develop and practice my teaching. And at the end of the day, I think that is the most important thing that a teacher can do.