Teaching Statement

Omar Ibrahim

My main teaching experience has been with introductory computer science classes where I have had an intimate view of how students learn at the threshold of computer science, and with discrete math courses, where I have helped ensure that our courses are able to best serve all of our students.

Making Students Feel Comfortable

Computer science *should be* a field that is open and welcoming to all, but historically it hasn't been. A vital part of changing that is recognizing that much of what we know about teaching computer science is tailored to people who are already through the door, and we need to make those doors accessible to all. Students don't come to us with a neutral view of computer science and we are often fighting an uphill battle both to attract and retain students in a field that is actively or passively not built for them.

Getting students in the door is one thing, but making sure that once they choose to study CS they can thrive is just as important. Students' success in my classroom should be based solely on what they do in my classroom; never on their prior experience or their cultural or linguistic background. It's important to me to be responsive to student feedback so that I can adjust my teaching methods to best suit my students -- not every teaching strategy will work for everyone. This is especially important when considering how to make CS classrooms more comfortable for students from historically underrepresented minority groups. I make sure to actively and vocally respond to feedback in some way every time I take feedback, even if it doesn't require changes to my course or teaching. This way, students see that I take their suggestions seriously, and feel more comfortable offering sincere feedback to me. Another practice to make students more comfortable is the "think-pair-share": students think through an answer to a question on their own, then discuss answers with their classmates, and finally share answers with the class. It gives every student a way to speak their mind without having to present something to the whole class. Discussing their answers before sharing can also give students who otherwise might not have felt comfortable offering an answer confidence in their answers and encouragement to participate more actively in class.

Communicating Transparently with Students

Transparent communication is important to serving our students as best we can. Sometimes, classes have hidden learning objectives; things that students are expected to take away from a class without explicitly being told that that is an objective of the class. I value objectives that go beyond learning new material, and I prefer to communicate these objectives to my students. I often find that when students are given tasks that serve an objective they are not aware of, they have trouble motivating why they're doing that work. For instance, in a workshop course for discrete mathematics, students were assigned to form study groups with other students in the class. From the perspective that our course is solely about discrete math skills, there's little

motivation to commit to being a part of a study group. However, one of the objectives of the workshop course was to help students build study skills and support networks. When we communicate that explicitly to students, they tend to be much more receptive to spending time working on *those* objectives.

Taking the Fear Out of Learning

Educational spaces can be scary for students, and that fear can be detrimental to students' learning. Low-stakes formative assessments can be incredibly valuable for students as ways to gauge their understanding in a low-stress environment. Having a frequent gauge for where you as a student are at is very helpful, even if that gauge is telling you that you don't yet understand something. It's good to know what you don't know. However, students are unlikely to genuinely engage with these kinds of assessments when "not knowing" might have disastrous consequences for their grades. I've incorporated low-stakes check-in assignments in all my classes with great outcomes. In my discrete math course, we gave weekly Quick Check assignments which were given full feedback on the quality of work but were graded on effort and completion. The low-stakes feedback was so valuable to students that most completed every Quick Check despite not being required to. Low stakes also allow us to provide more tools to help students; after completing the assignments, students were given an explanation of the solution to which they also had to respond. Students still regularly submitted incorrect work, and no one ever copied the given solutions. Low-stakes assignments help students value the learning process and feedback, not just getting the right answer on the first try.

Discrete math can be an intimidating course, especially for students who have only taken programming-based classes. One way we can make intimidating courses more approachable is to leverage the background knowledge that students come into our courses with. Building connections to what students already know helps lower the cognitive load we put on students. When I taught a discrete math course, I made sure to build those connections to programming concepts they were already familiar with to provide students with something to anchor new knowledge to, and an easier time extending that knowledge in the future.

Helping Students Help Themselves

Students require a certain level of exposure to new concepts to be able to understand them, and then more to be able to move on to apply, analyze, and evaluate their use of them. We only have so much time with students, so inevitably some of that exposure happens when we're not around. Accepting that inevitability allows us to build tools to make that easier for students. In my classrooms, I have built tools that provide automated, expert feedback when students are working on their own. A big part of this is breaking topics down and determining what forms of feedback/practice can be provided to students without direct intervention. Even proof-writing can be split into exercises with automated feedback. Providing students with more ways to practice, particularly in a guided way, is a great way to help students progress their skills, and finding a way to provide that guidance without having to be present is vital, particularly as we move towards more and more large-scale computing courses.