Teaching Philosophy

Andrew Rosen

I consider teaching Computer Science to be the most enjoyable and rewarding thing I do. I have won a departmental award for my teaching methods and consider myself exceptionally good at it. This is because I use a variety of methods to engage with my students and address the challenges computing.

Computer Science is a large field with many applications and that makes it particularly exciting both to learn and teach. I often tell students that I chose Computer Science for my undergraduate degree because I had no idea what I wanted to do, but I knew that Computer Science and programming would give me the power to choose any field I wanted. This is why I love teaching Computer Science: I can engage with a wide variety of students, each with unique backgrounds and widely varying aspirations.

With this shear breadth comes a challenge - Computer Science can be very difficult to teach and for students to learn. I posit three reasons for this.

The first factor is that Computer Science is a completely new and foreign field for the vast majority of students. Where many degrees teach a system of theory or a practical craft of programming, Computer Science requires both. This means we demand a great deal more from our students, essentially a complete immersion in the field.

The second is stringent requirements of assignments. There is no other subject where the expectation is that homework is completely correct every single time. A single typo can be the difference between resounding success and utter failure. This can be quite intimidating to the novice programmer.

The third is that you are your own worst enemy when you program. A single misplaced brace or missing equals sign can cause hours of mind-numbingly agonizing debugging. Even the most experienced programmer is not immune to this; everyone makes mistakes.

I teach using a wide variety of approaches to address these challenges. Often times, while the curriculum is solid, the materials provided by textbook companies (such as slides and test banks) are woefully insufficient. I create my own teaching aids, and make them available to the class. These teaching aids include my lecture notes, practice exams, code I write in class and prepare before hand, and topic-specific video lectures. This removes much of the fear and trepidation of the unknown that can get in the way of a student learning.

I believe the best way to learn programming but by actually programming. Reading slides is not sufficient to teach students, especially in lower level courses. By programming in class and prompting students for input, I am able to use the lecture time to get students coding, albeit vicariously. It makes programming feel more real and less theoretical.

Live-coding means making mistakes, and these mistakes, intentional or unintentional, help students become accustomed to the debugging process they will have to do themselves. I believe that seeing an ostensible expert make mistakes and correct them makes coding much less intimidating. As the course progresses, students become more comfortable with correcting any mistakes they see.

I test how well my students understand content and how they can apply it, not how well they can memorize information. My exams are open note to reflect the reality that professorial programming is almost never done without reference material readily available. This also allows me focus on testing my students abilities to apply information, not just recall it.

I also make myself readily available to any student who needs help. In many cases, when students are unable to make it to office hours or be on campus, I have met with students virtually via a video chat in order to help them understand the material. I have seen many instances where the subject matter "clicks" as a result of meeting one-on-one.

While I believe in a highly practical education, no Computer Science student can be as successful as possible without a rigorous theoretical understanding of the mathematics that we rely upon. I draw upon both math that students will need, but also make my lessons as topical and broadly applicable as possible.

For example, when I created an assignment to teach about sources of network latency, the Mars Discovery rover had just landed. The assignment had students calculate the delay in sending information back to Earth from the Moon and from Mars. Another example is how I teach students how to solve

complex math problems with programming by live-coding Project Euler. I weave history into my lessons when I introduce a new subject and show how this tiny sliver of Computer Science grew and impacted the world.

How can I be sure that my teaching methods are effective? One of my markers for student success is seeing students being confident enough to ask and answer questions in class. It is at this point that students independently ask and suggest how what they learn can be directly applied to one problem or another. When this happens, the student has learned the material to the point they can apply it outside the context of a classroom. I know I am successful when I see that my classroom is full of people who have shown up, not because they have a course assigned at this time, but because they are there to learn.

What is it like to be my student? I teach that Computer Science may look hard, but that is only because it is new. I teach that what you learn in my classroom is broadly applicable, especially in the sciences. I teach you that it is always appropriate to ask for assistance when you do not understand something. I teach you to not be afraid to be wrong and learn from mistakes and to always ask questions. I will show you that programming can not be learned by just reading books, but by practice, and I will give you many opportunities to practice.