I enjoy teaching, and the chance to teach is my primary motivation for seeking an academic job. I've always sought out the opportunity to teach: my first job was teaching kids LOGO, and I have been a TA throughout my undergraduate and graduate career. Below, I'll describe some of my teaching experience and then briefly explain my teaching philosophy.

Teaching experience. I have been a TA at every level of university education:

- graduate classes for PhD students (1x: program analysis)
- graduate classes for part-time, working master's students (1x: testing and debugging)
- senior-level undergraduate electives (5x: software engineering, programming languages, game design)
- core undergraduate classes (6x: data structures, intro to CS)

In these TA roles I've experienced most parts of teaching: I've designed and given lectures, designed and graded homework problems, designed and graded exam questions, designed slide sets for TA-led discussion sections, managed other TAs, held office hours, answered student questions online, designed and maintained course infrastructure, etc. I've also taught an undergraduate seminar that I designed myself. Teaching is fun, and I don't know a more rewarding feeling than seeing a student I've taught go on to succeed.

Teaching philosophy. When I'm teaching, I have four core priorities: choosing the right implicit curricula, giving fast feedback to students, balancing consistency and fairness, and using enthusiasm to engage students.

Implicit curricula. A course should not just be a set of facts presented by a lecturer; it should also have an implicit curriculum to develop the skills of the students. In my experience as a TA, the best instructors I worked for always knew the implicit curricula of their classes and shared it with us. For example, in an undergraduate data structures class it is a disservice to the students to *only* teach the implementation details of the data structures. Students perhaps should also develop their ability to understand why code is written the way that it is: a skill that they will need in more advanced courses and in their careers. Courses—even with ostensibly similar curricula—with different implicit curricula demand different designs. An undergraduate data structures class whose implicit curriculum is learning to read code written by others must be taught differently than one whose implicit curriculum is deeply understanding recursion. When I teach, my first step is to identify the course's implicit curriculum within the context of a student's overall education. Especially in undergraduate education, where reusing teaching materials between instructors is common, it is easy to forget to think about the implicit curriculum—but being aware of it is half the battle.

Fast feedback. When training a dog, if you don't catch them doing something bad while they're actively doing it, disciplining them later won't teach them anything or change their behavior in the future. The same principle also applies to people: students learn better when their mistakes are corrected as quickly as possible. This observation should pervade course design: homework assignments should be short so that TAs can grade them within a day or two, exam problems should be designed like a proof produced by a verification tool—hard to find the solution but easy to check it—and automated real-time feedback should be used whenever possible (programming assignments often have this property). My enthusiasm for this idea comes from seeing it work well in my undergraduate compilers class, whose final assignment was to build an optimizing compiler for a toy language. A web dashboard showed which compiler produced the fastest code for the benchmarks. Students submitted anonymously (important not to discourage students), but the same profiling tests were used to determine the students' grades, so students always knew where they stood.

Consistency and fairness. An instructor should try to achieve both *consistency*—treating all students the same—and *fairness*—rewarding students proportionately to their learning. Consistency and fairness can be in tension: consistently grading a student who missed a significant assessment due to illness would be unfair to that student, but giving that student an easier path to a good grade would be inconsistent. An instructor who fails to balance these two demands can discourage or disillusion students. Further, I believe that the appropriate balance is a spectrum that changes with the course. A core undergraduate class taken by hundreds of students should emphasize consistency because the instructor cannot fairly evaluate the circumstances of hundreds of individual students, which might lead to bias. A class for PhD students working on individual projects related to their research should emphasize fairness so that each student can succeed both in their own research and in the class, according to the criteria that matter in their work.

Engagement. The preceding philosophical ideals are primarily about course design and organization. But for these things to matter, students have to want to be in a class. For this reason, I work hard to be an enthusiastic and energetic lecturer, and to actively encourage student participation. I've found that students respond well to simple manipulations, especially when they know they're being manipulated: for example, when I want students to ask questions in class, telling them that I'll give them a piece of candy the first time they ask a question works well as an inducement. They know that it's a bribe—and yet they still participate more because of it.