The main guiding principle of my teaching philosophy can be summarized with a quote from Prof. Caroll Morgan:

To put new knowledge into students' heads,
you must first create a void for that knowledge to fill.

As a student, I had many opportunities to experience the importance of motivation in the learning process. One of the most memorable was a graduate course *Formal Methods for Information Security*. In our first homework we were given a description of a cryptographic key exchange protocol that contained a vulnerability, and were asked to find what could go wrong. Doing this by hand was painfully difficult, and I was never sure I found all the problems. Later in the semester we learned about verification techniques capable of exposing all the vulnerabilities automatically; this highly-technical material appeared straightforward and natural, because it was presented as a solution to a puzzle.

**Teaching experience.** As a graduate student at ETH Zurich I was involved in teaching almost every semester as an assistant for *Introduction to Programming*, *Java and C# in Depth*, and *Software Architecture*. *Introduction to Programming*—a freshman class with about 300 students—was an especially interesting and challenging experience. In addition to basic programming skills, it introduced students to *Design by Contract*: a programming methodology based on rigorous reasoning about program correctness, using pre- and post-conditions, and (loop and data-structure) invariants. I was thrilled to see that these concepts not only helped students improve the quality of their code, but also drastically simplified the presentation of some nontrivial algorithms and data structures. This experience made me an enthusiastic supporter of introducing elements of formal verification into undergraduate programming classes.

The biggest challenge in *Introduction to Programming* was the diversity of the students' background: some of the incoming freshmen had never written a line of code before, while others had had several years of industry experience. During the six years that I was involved in the course, I worked as part of a team to evolve the material and make it both accessible and engaging for all students. For example, we split the recitation sections into three different levels depending on the students' self-assessed level of prior experience. This way beginners could get more guidance, while advanced students focused on topics like *Design by Contract*, which are not a common practice in industry.

In a lecture setting, I really like interleaving theoretical material with live tool demonstrations and hands-on exercises that students and I collectively solve in class. I used this style in my guest lectures for *Software Verification* at ETH and *Foundations of Program Analysis* at MIT, as well as two tutorials on program verification and synthesis tools that I gave at the annual MIT Programming Languages Offsite in 2015 and 2016. I found this to be a great way to build a feedback loop between theory and practice and keep the audience motivated and engaged.

**Mentoring.** I have always enjoyed working with younger students in a mentoring capacity. As a graduate student and a postdoc, I supervised three undergraduate research projects and three master's theses (one of my master's students later received a PhD degree from EPFL). This experience made me realize the importance of finding a good balance between letting students pursue their own ideas and giving them the guidance they need to succeed, in the form of weekly discussions, intermediate deadlines, or a reading list. I also realized that every student is different, and a single style of guidance does not fit all. At MIT, I also had a very rewarding experience working with a younger PhD student, who led her own research agenda but needed help with refining the presentation of her work for a publication.

**Future Teaching.** I would enjoy teaching both undergraduate and graduate courses on programming languages, compilers, formal methods, and software engineering. In addition, I am interested in contributing to introductory programming or algorithms and data structures courses, and in particular, bringing elements of formal reasoning about program correctness into those courses. I would also be happy to teach undergraduate classes outside my immediate areas of interest.

At the graduate level, I would like to teach classes on program verification, program synthesis and repair, and SAT and SMT solvers. I believe that my extensive research experience in several areas of program verification puts me in a position to design a course that covers a wide range of topics—from bounded model checking and abstract interpretation to interactive and automated proof assistants—and presents the diverse concepts behind these techniques in a unified and systematic way.