Srinivas Narayana: Teaching and Mentoring Philosophy

To me, teaching computer science (CS) is about teaching a set of concepts and cognitive tools that are both intellectually stimulating and help solve real-world problems. Students come to CS programs from diverse backgrounds, from curious beginners to those who have immersed themselves for years. I believe that everyone can learn the tools of computing and use them to solve problems effectively.

I view teaching broadly to include classroom instruction, mentoring through research collaborations, and even giving research talks. I am very fortunate to have had several opportunities to teach. As part of India’s National Service Scheme (NSS), I taught mathematics to middle school children at St. Louis Institute for the deaf and the blind, in Chennai, India. In my senior year at IIT Madras, I conducted a hands-on networking tutorial, where I demonstrated tools such as wireshark, browser extensions, and curl to a classroom of about 100 students from all over India. At Princeton, I taught as a lecturer for software projects in Operating Systems. I also had the opportunity to craft programming assignments and to mentor team projects for an advanced programming class. I have mentored a number of undergraduate and graduate students both at Princeton (Michael Chang, Vibhaa Sivaraman, Mina Tahmasbi, Anshuman Mohan) and at MIT (Akshay Narayan, Prateesh Goyal), co-authoring multiple research papers with them. The undergraduates I mentored—Vibhaa (Massachusetts Institute of Technology), Michael (University of California at Berkeley), and Anshuman (National University of Singapore)—are all currently pursuing graduate-level research. I volunteered to teach at a middle school in Boston during Boston’s STEM week, introducing them to computer programming. Most recently, I completed the Kaufman Teaching Certificate Program (KTCP), a teacher training workshop at MIT [5]. The KTCP included sessions on designing courses, helping students learn, engaging students, assessing effectively, and teaching inclusively. I also taught a small lecture in front of the course instructor and peers who later provided feedback. I will now describe my teaching philosophy and how my experiences have shaped it.

Relating computing to the real world. Concepts from CS are all around us:

- Why is it easy to look up a word in a dictionary?
- How do you know if a letter you wrote has reached its recipient?
- What’s the most efficient way to tell when your fast-food takeaway has arrived?
- What’s the most efficient way to get to the right water temperature when you shower?
- How did you solve paper mazes? How do you find the best route between two buildings on campus?
- Why does using a washer and dryer simultaneously reduce total laundry time?
- Can you relieve the road congestion at 8 AM for everyone on campus?
- How can we understand each other while conversing in a noisy room full of people?

I will use such examples and questions, and come up with new ones, to introduce concepts like hashing, reliable communication, information coding, feedback control, interrupt processing, pipelining, optimization, and so on. From my experience teaching Operating Systems, I learned that using examples and analogies that students can relate to (e.g., “context switching” as the transition between school work and hobbies) significantly improves their attention. During Boston STEM week, I had the pleasure of introducing programming to middle schoolers using a tool that draws art on the screen through a user-specified context free grammar [1]. There is evidence from pedagogic research that connecting abstract and concrete representations of concepts enables better learning outcomes for students overall [3]. By providing a scaffold to hoist CS concepts in the real world, students of diverse backgrounds and experiences can learn equally well, in their own ways, and generalize their learning to other settings more easily.

Building artifacts together with people. I have repeatedly been inspired by teams of people producing creative artifacts together—both from my teaching experiences at Princeton and my research collaborations. I will incorporate projects in my courses where students work in groups for a part of the semester, defining a problem they care about, and building prototype systems that they can use themselves. This is likely to work well in a senior or graduate-level class (but suitably scaled-down versions can work at other levels too). I have observed that students are highly motivated and perform their best when they are personally invested in the fruits of their own work. Through my teaching and research mentoring, I will help students
• think through problems at multiple levels of abstraction,
• think modularly by breaking complex problems down into independent subproblems,
• generalize specific solutions to broader problems,
• learn to start with imperfect solutions and iteratively improve them,
• learn to build on the work of others,
• learn to work with others with different backgrounds,
• learn to communicate their questions and results (even if incomplete) to others,
• learn to work honestly and collaboratively, and
• make their own work and artifacts useful and reproducible for others.

Many of the abilities I have listed above form the basis of the so-called “computational thinking” \[2\] traits. I plan to meet with students at a reasonable cadence (e.g., 1–2 weeks) to provide early and frequent feedback on their learning and progress. Such meetings have enhanced both my research and the software projects I mentored at Princeton.

**Applying well-researched pedagogic methods.** At the KTCP workshops at MIT, I was exposed to a variety of teaching techniques which are well supported by pedagogic research. In particular, active learning methods \[6\] help students learn more effectively by interacting with the material, the instructor, and with each other in class. I will try the following active learning learning methods, besides others, in my classroom.

1) Practicing retrieval by completing details: When I taught lectures in Operating Systems, I suppressed certain details in slides, instead asking students to complete them in class \[4\]. For example, what is the processor state that is unique to a process on a computer? What does it take to provide the abstraction of a dedicated machine to a process? How do you design a data structure that provides desirable properties on the tasks scheduled? Such questioning made the details accessible to the class as a whole, boosting students’ understanding of concepts. When students noticed that the barrier to answering a question is low (e.g., it is enough to suggest one point; not an entire answer), almost all students in my classroom were highly likely to engage. I was also very satisfied with the students’ project scores.

2) Answering deep explanatory questions: I will have students compare multiple approaches to a problem, for example: should we let intermediate network nodes manage reliable message communication in a network, or just the two communicating ends? Is it better for a packet transmission algorithm to wait until a loss, or increased delays, to reduce its sending rate? There is strong evidence that deep explanatory questions promote high-quality student learning of course material \[3\]. CS is abundant with opportunities for such questions. I hope to make it fun for students to answer, say, by running a “wheel of fortune” animation in front of the class to pick the next student who answers.

3) Maintaining a journal: I will have students write “reflection” journals and put it up on a web page where I can read them. (I can help set up a web page if needed.) Students can write anything that is course-related: a concept that intrigued them, a question about the day’s lecture, or an obstacle in an assignment. I will allocate a participation grade for this journal, so that every student must reflect on the things they learned at least once a week. By having students reflect on material taught in the last few lectures, their recollection of material is spaced out over time, and learning is more effective \[3\]. The journal also allows students (especially underrepresented groups) to seek help freely, without stereotype threat. I have myself written a journal for a class at IIT Madras, and found it to be very effective.

In consultation with students and faculty, I intend to refine my teaching methods over time. From my teaching experiences ranging from middle school to graduate school, I have realized that effective teacher-student communication depends on understanding the preconceptions, motivations, and learning abilities of students—qualities that vary widely across classrooms and students, and are not necessarily known ahead of time. I hope to tailor the student learning experience as I learn with and about my students, with the good faith that students will get excited about learning and innovating in CS, both for its intellectual rewards and to provide tangible utility to society.

**Teaching plans.** I can teach the undergraduate-level Networking and Operating Systems classes, and handle freshman and sophomore-level introductory CS classes. I could also teach a graduate-level Networking Seminar where I would discuss research papers in the area and support students working on projects over one to two months. I would also like to develop a ‘special-topic’ graduate-level class on Programmability in Computer Networks, which would cover the abstractions and the conceptual tools to make networks programmable (language design, hardware design, compilation techniques, testing, verification, synthesis), with readings of relevant research papers. Typical course projects may improve the programming expressiveness, performance, or correctness of switches and endpoint software stacks.
References


