Learning computer science (CS) can be both exciting and challenging. Perhaps, a major problem a CS student faces is *information overload*. With so much to learn and so much of information (and misinformation) easily available, learning CS concepts, staying in the *flow channel* (neither getting bored nor panicking), can be extremely difficult.

As a teacher and mentor, my key objective would be to show students that learning CS is about grasping a few fundamental principles and intuitions that underlie a plethora of application-level technologies. For example, in a *programming* course, my key objective will be to help students learn the art of breaking down a solution to a problem into a *set of instructions*; the syntactic nuances of a specific language come afterwards.

Interests. At a foundational level, I am interested in teaching *software engineering*, *data mining*, and *empirical methods*. As these subjects are core to my research, I have been studying and reflecting on them for several years. At an advanced level, I am interested in teaching subjects that lie at the intersection of traditional subjects, specifically, *social computing* and *data-driven software engineering*. Whereas I will focus on fundamental concepts in the introductory courses, I will introduce challenging research problems in the advanced courses. Finally, being an avid and experienced software developer, I also enjoy teaching *programming*, *algorithms*, and *data structures*.

Experience. I served as a *teaching assistant* (TA) for three courses: *service-oriented computing* (four times), *social computing* (twice), and *graph theory* (once). In addition to holding office hours (to clarify students' doubts), and grading homeworks and exams, a key responsibility I handled as a TA was to design programming assignments. I designed these assignments such that they (1) reinforce concepts taught in the lectures, (2) train students on state-of-the-art technologies, and (3) reduce the potential for plagiarism (by designing each semester's assignments to be sufficiently different from previous semesters'). Although time consuming, this effort felt worthwhile when I heard from several students that these assignments played an important role in fetching them industry jobs.

Next, working closely with three junior PhD students in our lab, I got a glimpse of *mentoring* experience. I helped these students in shaping their ideas as research problems, designing experiments to validate those ideas, and most importantly, presenting the ideas and experimental results in research papers. Some of these collaborations have resulted in high-quality publications and some are ongoing.

Methods. First, I plan to incorporate *conceptual modeling*, an idea I explore in research, in teaching, too. Conceptual modeling advocates that a software (to be implemented) be understood via cognitive concepts such as users, and their goals and plans, to start with. The crux of the idea is to focus not only on *what* a software should do and *how*, but also on *why* the software should do so. Following this intuition, for example, when teaching *principal component analysis* (PCA) in *data mining*, I will not only describe how to implement PCA, but also describe that it works (reduces dimensionality) because the principal components are uncorrelated, in contrast to the original set of potentially correlated variables.

Second, I consider balancing theory and practice as an important aspect of teaching CS. I will design my courses to educate students on the foundational concepts as well as to train them on using those concepts in concrete applications. This will both develop students' intellect and equip them with the skills their careers demand. Learning from my TA experience, I will make sure that the training exercises reinforce, but also be complementary to the material I teach in lectures. For example, if I teach *version control* in *software engineering*, I will design an assignment asking students to map *Git* (a contemporary version control system) commands to the concepts taught in the lecture.

Third, higher-education institutes are culturally diverse. Whereas diversity enhances students' experience, it makes teaching a challenging task. For example, I believe in engaging students via dialogue during a class. However, this may not be easy: whereas students from western countries tend to be outspoken, Asian students tend to be shy (some might even consider disagreeing with the instructor as disrespectful). A solution in this case can be to encourage students to first talk to their peers and then to the instructor. I have experienced many such cultural differences in my long student career. I respect such differences and will do my best to accommodate for them in my classes.

Fourth, students often do not have a bigger picture of CS careers, e.g., industry vs. academic jobs or development vs. testing in the industry. I am willing to help students understand various career options and prepare accordingly. For example, a student wishing to pursue a development job in the industry must master programming, whereas publishing a paper (or even attempting) can add a great deal of value to a student wishing to pursue a research career.

Finally, motivation is important for learning. I imagine that a student does well in a subject not just because of dedication, but also because he or she is passionate about the subject. To inspire students in my courses, I will demonstrate how the concepts they learn could lead to applications that benefit millions of users. For example, when teaching *artificial intelligence*, I will describe how popular applications such as *Watson* and *Siri* employ advanced versions of some of the search algorithms students learn in the class. Similarly, in advanced courses, I will invite researchers to present cutting-edge works relevant to the course to inspire students about research careers.