Teaching Philosophy
Guni Sharon

When I was a young Ph.D. student, I was requested to meet with potential undergraduate students attending the open day at Ben-Gurion University. My task was to have personal chats with the candidates and answer their questions. To my surprise the question that came up the most was “can I earn a lot of money with this degree?” I was surprised by that question since I always considered higher education as a goal, something to strive for, but on that day I learned that many young, soon to be students, see it as a means. I believe that such a perspective is counterproductive to the central goal of higher education which is training individuals to have comprehensive knowledge and wide perspective. As I see it, educators should provide their students not only with knowledge but also an understanding of the value of knowledge. Once students understand that the value of obtaining new knowledge far exceeds that of a good salary, they will be ready to partake in substantial academic training.

When teaching in front of a classroom I believe that a good instructor should avoid focusing on the solution of a given problem but instead focus on the solving process. Together with the students, during class, I usually discuss ways to analyze and approach each problem. I believe that showing and discussing wrong ways to approach a problem is almost as important as showing the right approach. For example, when showing students how to devise a finite automata for a given problem, I begin with a clean board and ask students to intuitively suggest ways to proceed. I usually focus on students’ suggestions that seem, intuitively, correct but that I know to be wrong. After the students realize that we are heading in the wrong direction, I explain to them why their intuition in this case is wrong and where they should focus their attention. I found that many students reach a much deeper understanding after discussing common mistakes. I take a similar approach when instructing graduate students during research projects. Allowing them to make some mistakes proves beneficial to their overall, in depth, understanding of a larger picture.

During my time as a teacher assistant at Ben-Gurion University and as a postdoc at the University of Texas, I guided many undergraduate students in different assignments and research projects. In many cases, I was surprised by the lack of basic programming skills that the students exhibited. For instance, while guiding an undergraduate student in a research course at UT (CS370), I noticed that the code he produced was badly designed, mainly due to lack of knowledge and programming experience. The student’s code contained many repetitions due to poor use of inheritance and generic functionality and also misuse of data structures resulted in inefficient memory consumption and CPU runtime. I believe that current approaches for teaching the basics of programming should be reconsidered. In my vision, the basic programming courses (Intro to Computer Science, Software Design, Data Structures, Software Engineering) should be much more “hands on”, where students learn, not only the theory of coding, but also partake in intense, large scale, coding projects. Starting from a very simplistic code design, I would have students go on a journey where, as they learn the basics of correct design and gain experience, the number of required lines of code diminishes, the readability improves, the debugging becomes easier and the overall efficiency grows. When teaching the Data Structures course, for instance, I would start by asking each student to implement a simple list structure for integers. During the following weeks I would ask the students to add more functionality and generic properties to each others’ projects (randomly shuffling the projects). By doing so I would teach the students the benefits of correct design for a continuously evolving

---

1I served as a teacher assistant for the “automata and computability Theory” course at Ben-Gurion University.
projects as well as the importance of standardized design for code readability between different collaborators. I find it unacceptable that a CS graduate of any university is not capable of writing well structured code.

Personally, I feel comfortable to teach any of the following courses: Data Structures, Algorithms, Introduction to Programming, Intro to Artificial Intelligence, Automata, Computability, and Complexity theory. I would also feel comfortable teaching electives courses on multiagent systems and heuristic search. I have both teaching and research experience that is relevant to each of these course topics. I will seek for inspiration, when structuring a new course, in slides used by previous instructors from inside and outside of the department. I will then explore for latest breakthroughs and trends that ought to be added to the course’s material. I plan to define personal assignments that would require students to think out of the box and self-explore for solutions beyond what is presented in the class.

Overall, I see myself as a patient, demanding, and fair instructor. I try to engage my students during lectures and encourage them to become involved in a thinking process where, together, we approach a given problem. Through my teaching and mentoring experiences, I have become more confident and have a specific desire to teach. I also look forward to mentoring graduate students, teaching assigned classes, and developing my own classes. I love to teach and am always looking for new and more advanced ways to challenge my students to think outside the box and become good problem solvers.