TEACHING STATEMENT

Kangkook Jee (kjee@nec-labs.com)

Passion for teaching is one of the primary reasons I would like to return to academia, and I have a strong desire to communicate this positive synergy with students. I believe that our knowledge grows when it is shared with others. For this reason, unlike other graduate students who preferred to focus on academic publications, I voluntarily designed and taught an advanced programming course in my fourth year of graduate study at Columbia University, even though I had already met the teaching requirement.

Teaching experience

In Fall semester of 2013, I designed and taught a programing language course on Python language. Designing and preparing a course was a difficult task. Although I was already familiar with the language and course subject, organizing and transforming fragmented knowledge into coherent, presentable material took considerable effort. As a non-native English speaker, delivery was even more difficult.

However, my effort was generously rewarded when I received the student evaluations with a course rating of 4.45/5, which is considered a very high score at Columbia. The students mid-semester feedback had been not only encouraging but also very helpful, and substantially guided me to improve the course content and its delivery. The experience refreshed my strong interest in and the importance of communication and knowledge sharing as a way to educate myself as well as the students. Although the course was somewhat basic and I was very familiar with the material, I learned previously unseen details, and overlooked misconceptions. The experience reminded me of the old saying the educator himself is the most significant learner. My teaching experience has also taught me how to segment material, make new information interesting to students, and help them learn through hands-on experiences.

Python being an easy to use language with a variety of usage cases, at the beginning of the semester, I surveyed the students about the applications regarding their expectations for the course. I reflected student inputs to course design, and I tried to maintain this feedback loop throughout the semester. To keep students be interested, I also used materials movie clips that showcase a quite advanced but exciting use case of Python language and its packages. By actively engaging them in the course and giving them practical tasks that build on previous knowledge, students become more interested in the material, and they can feel the success of learning.

One-on-one Mentoring

Another form of interaction that I enjoy is one-on-one mentoring. Throughout my research career in graduate school and the years at NECLA as a researcher, I had the privilege of working with bright young talent from a variety of backgrounds.

In most cases, the one-on-one mentoring focused on delivering specific research outcomes, but the key principle that I have as a research mentor is that the mentor should let the mentee sit behind the wheel. The mentor is mostly needed when the project encounters difficult problems or the mentee comes across complicated situations. The mentor should help clear the roadblocks so that the mentees can progress smoothly.

In other words, the mentor should guide the mentee along the way instead of strongly directing the experience. By treating mentees as colleagues instead of as junior employees, mentees can feel free to come to me and discuss the challenges they face. Even when I supervised the interns, I tried to relate to them in person, not just as a resource to be expended for the project. I send a signal that I care about them individually, and I care more about their careers.

Many of these past mentor/mentee collaborations have produced fruitful outcomes including academic publications and project delivery as well as lasting friendships with these I talented collaborators. Likewise, I look forward to interacting and collaborating with diverse talented students at your institution.

Teaching Principles

The CS curriculum design at the university level needs to be balanced between practical skills and theories. Covering the latest technology will help students prepare for the future with skills required by todays market. However, this type of knowledge often tends to short-lived with the rapidly changing technology environment and it does not cultivate foundational traits.

Thus, to differentiate the university-level CS curriculum from job training materials that can be easily found on the Internet, I believe university CS level courses should provide the following.

- Primary and preliminary mastery of programming and an understanding of underlying runtime implementation.
- A solid understanding of a mathematical foundation for CS theories.
- The essential experience of dealing with open-ended problems.
- Encouraging critical thinking since a programmer is a problem solver by nature.

Example Courses

I look forward to teaching undergraduate and graduate computer science courses related to computer science fundamentals, systems, and system security. Although I am not limited to the following courses, these are basic courses I would enjoy teaching or even designing.

Basic programming course. I would enjoy designing and offering a basic / introductory course that starts from a primitive language such as C/C++ and makes a gradual transition to high-level dynamic languages (e.g., Python, Javascript, and Golang). With this balance of old and new programming paradigms, students can compare different aspects to better understand the usability and the efficiency trade-off and historical context behind their evolution.

Advanced programming course. An advanced course can first teach a procedural language (Java) and then move onto a functional language (Scala). From the course, students should not only learn new languages but also experience how these two languages of different paradigms achieve computational equivalence and its language theoretic background. The course should cover the runtime environment by introducing common intermediate representation (Java Bytecode), and its language runtime (JVM).

Operating Systems (OS) course. This course is a must-take course for most CS majors, since it is essential for understanding the primary operating system abstractions, mechanisms, and implementation. While most course materials can be illustrated with a Linux system, which is the de facto standard platform for many systems (*e.g.,* IoT OSes, Android), the course can add other mainstream OSes (OSX, Windows). In particular, Windows systems are of more interest to security researchers since it is most often used in real-world security warfare. Understanding the program execution model and its kernel structure is additional interesting knowledge to prepare students to be advanced system security professionals.

Advanced OS course. This course would mainly cover OS-level support for security and accountability. The course typically focuses on the latest OS features related to security (SE Linux, HMAC), isolation (support virtualization, container), and monitoring (performance counter, audit, kernel instrumentation).

System security course. As an occasional CTF (Capture-The-Flag) competition participant, I would like to design and participate in a security course that introduces the core concepts of system and network security challenges with CTF-style hands-on assignments and projects.

Reaching beyond the classroom. Extending the system security course, I would like to suggest an extracurricular activity to prepare a group of students for CTF competitions and train a group of highly skilled white-hackers. This type of resource is in high demand in industry but it is rarely taught. The realization of a hacker mindset will benefit not only students who seek a career in security but will also be useful for general CS students. CTF competition assumes that students have a thorough understanding and mastery of computer programs and its operational protocols so students can use their knowledge in a very practical competition.