Teaching Philosophy

"The point of the journey is not to arrive" –Neil Peart

I believe this quote more than any describes my teaching philosophy. Solving a single problem is great, but learning how to solve a problem is far more important. I have had the pleasure of having many teachers and mentors who valued the journey of arriving to an answer far more than the answer itself. In my opinion, becoming a strong computer scientist involves having a tool belt and understanding how to use it. To do this, you need to think critically and understand how each tool should be used. My goal as a instructor is not simply give students the tools they need to solve problems, but also the ability to pick up new ones and teach themselves how to use them. To accomplish this, I strive to create an environment that encourages curiosity, engages students, and feels safe.

Encouraging Curiosity

A healthy curiosity is crucial to building an understanding of a topic as complex as computer science. Topics build off of each other. Curiosity leads to discovery which is one of, if not the most, exciting thing in learning. Like reading a good book, students should look forward to new topics. I want them to theorize what comes next, think how things connect, and hypothesize the consequences of what they've learned.

I am a naturally curious person, and I let that show in class. In the course of teaching, there have been several times where the student asked a question I either didn't know or didn't have a well prepared answer for. When this happened, I would investigate the topic and start the next lecture addressing the question. The ultimate goal of this process is to get students to want to investigate themselves. Lectures need to be fluid so they do not feel rushed and students feel comfortable asking question that arise, even if they are tangential. That being said, its important to keep things on track. If a student is very curious about some idea that is off the beaten path, there is always office hours.

Engaging students

Engagement is a crucial step towards learning. An engaged student will listen, think about the material that is being presented to them, and answer questions when posed, either verbally or internally. Engagement starts with the instructor. In lecture, I try to be as animated as possible. When possible, I will add humor to a lecture, and I'm not afraid to have it be at my expense if I misspeak or have a typo on a slide. I am very fascinated and passionate in many aspects of computer science, and I believe this passion and interests can be infectious. Further, its important to meet students where they are by relating material to things they know using analogies and metaphors. I make sure to check in with the class frequently to see if they are grasping the concepts. If not, this can lead to a dialogue between the class and I to get them on the right track.

Engagement may start with the instructor, but it goes beyond simply lecturing. Lessons need to fit the different learning styles of the students. Not everyone learns the same, so its important to incorporate several approaches in lesson plans to help everyone. Besides lectures, I also use collaborative learning techniques with hands on problem solving. When teaching CS1010, I would often create problems or pose discussion questions and have students get into small groups to work on them. By working in groups, often randomly assigned, students see different perspectives and learn through explaining. Explaining a problem forces students to think fundamentally about it and can show them the gaps in their understanding. During the coding portions of the course, I would set aside certain classes as labs where students could work together on assignments.

Building a Safe Classroom

To support the two prior points, students need to feel academically safe. By this, I mean they need to feel that they can talk through problems in front of myself and other students and not feel they need to be perfect. Students will not want to answer or ask questions if they feel they will be judged by them. This will lower engagement and hamper curiosity.

The instructor sets the tone for the course, so I try very hard to listen to students and understand what they are saying. If they are answering a question, sometimes their understanding may be a bit off. I try to pull truth

out of their answer or coax them towards the right one if they are close. Students need to feel confidence, and both of these techniques help prevent them being overly self-critical. Further, I show that I too am still learning. As I said earlier, I am not afraid to admit I do not know something, but don't I see this as a problem. Instead, it is an opportunity to investigate further.

Teaching Experience

Experience is very important to learning any skill. You need to do things to learn what you do well and how you can improve. I knew I wanted to teach students when I entered graduate school, and I sought out any opportunity I could to gain experience.

Teaching Assistant

I was the teaching assistant (TA) for the undergraduate compilers course and both the undergraduate and graduate level software analysis course at the University of Virginia. Beyond the normal TA duties, I gave guest lectures, helped develop assignments (including the course project), and assisted in exam writing. I took my role as a TA seriously and tried to learn as much as I could from the course instructors. I believe it is this effort and work that led two separate faculty members to nominate me for the "Outstanding Graduate Teaching Assistant Award", which I received. This experience prepared me to teach a course on my own.

Primary Instructor

After expressing interest in being the primary instructor for a course, I was given the opportunity to teach "CS1010: Introduction to Information Technology"[1]. This course is restricted to students outside of the engineering department and is meant to introduce computer science to students and prepare them to use computing solutions to problems they may face in their future endeavors. For this course, I produced brand new materials, creating new slides, assignments, an exam, and two final project students could choose between.

Being able to teach a course fully and build it from the ground up was an amazing experience. First, it confirmed that teaching was the profession for me. It was incredibly fulfilling and brought me a lot of joy. Second, I learned about what it means to teach a diverse body of students. Since the only prerequisite for the course was that student couldn't be in the school of engineering, I had students from very different backgrounds, academic and otherwise, and with separate reasons for taking the course.

I began the course with a survey so I could understand what they wanted to get from the course and where their level of understanding began. I intentionally left some lectures blank at the start of the semester so I could integrate the topics students where most interested in. Some students had preliminary computer science experience and others were brand new to the subject. As a result, lectures had to keep the attention of those with experience, but not be so complex that the rest were lost or intimidated. This was a tight rope to walk, but I believe I did it well, as evidenced by my course evaluations[2].

In this course, I was able to put my philosophy into practice with the following activity. Early in the course, I wanted to give the students an appreciation for the power of computers. To do this, I put students in pairs and gave them all a piece of text which had been encoded using a Caesar cipher. The decoded message was "Face the back wall". I explained to them how a Caesar cipher works and told them the text was an instruction they should follow. As time went on, some students solved the puzzle and they turned around. As more students did this, those that remained realized roughly what the text said, and they used this to help them crack the code. It took students somewhere between 5-15 minutes to solve the cipher. I then loaded up a computer program and it solved the problem in a fraction of a second. To show them it wasn't a trick, I had them give me text to cipher and de-cipher. A few weeks into the semester, I asked for feedback and several students pointed to this as an engaging and interesting activity.

The following semester, I taught CS1010 again, allowing me to further improve my teaching skills. Between semesters, I was able to reflect on the course and revise it according to what went well and what needed

[1] Course website located here: `https://will-leeson.github.io/CS1010/`

[2] Course Evaluations: `https://will-leeson.github.io/assets/pdf/CS1010-Teaching_evals.pdf`

improvement. I spent time reading papers on research-based instruction strategies which I could integrate into the course. From the course evaluations, it was clear students valued active learning activities, such as think-pair-share, cooperative learning, and problem-based learning. To accommodate the students learning styles, I altered the course to spend less time lecturing, and more time where the students are active participants in each session. This was an interesting challenge as I had to ensure what lecturing I did covered material that could inform the students ability to do activities. It also caused me to think about what the students could learn organically through activities and what required preliminary information in the form of lecture.

Teaching Interests

While I believe I am capable of teaching any undergraduate computer science course given enough lead time, I would be most excited to teach classes on the following topics:

- Introductory Computer Science
- Software Engineering and Formal Methods
- Theory of Computation
- Programming Languages and Compilers

Further, I would be interested in developing a program analysis course. Students are often taught how to debug and write individual test for programs. However, there are many powerful tools and techniques rooted in formal methods which can ensure systems act correctly. In the age of generative AI models, coding will become more and more automated, so the responsibilities of developers will likely shift from writing code to ensuring code acts correctly. A course on program analysis will give students a fundamental understanding of what it means for code to be “correct”, and how to evaluate it.