Teaching Statement

Jacob Neumann

Teaching has been a central component of my academic career—from the beginning of my undergraduate to my present position as a PhD student—and I am always looking forward to new opportunities to share knowledge with students. I find that teaching is both a means by which I can contribute to my university community, as well as a vessel for my utmost creativity and passion. I take great satisfaction in sharing my sense of wonder on a variety of topics and in seeing my students experience the same wonder, challenges, growth, and accomplishment.

Teaching Experience

I have been involved with a dozen different courses—as a tutor, grader, teaching assistant, and lecturer—and have had around seven hundred hours of student contact to date. I started my teaching career working for Carnegie Mellon University's Academic Development Center,[1] where I was trained in collaborative learning practices in order to provide supplemental tutoring for mathematics courses. My main teaching experience during my undergraduate was as a teaching assistant in the School of Computer Science, for a first-year functional programming course. This role included regular student contact, in both a group setting (recitation) and one-on-one (office hours). The weekly homework assignments for this course were designed and maintained by the teaching assistants, so I also gained valuable experience in designing and deploying assessments (as well as the practicalities of grading and providing useful feedback to students, including "auto-grading" huge amounts of student code by computer). In my later semesters, I held the role of 'Head Teaching Assistant', where I had additional responsibilities including interviewing and selecting teaching assistants, coordinating the activities of the course staff, scheduling office hours, overseeing the creation of new homework problems, and running weekly staff meetings. In my year as a master's student at CMU, in my gap year during the pandemic, and as a PhD student at the University of Nottingham, I have also served as teaching assistant[2] for courses in philosophy, formal logic, abstract mathematics, programming, and theoretical computer science, giving me experience teaching a wider variety of topics in a more diverse set of circumstances.

My most significant responsibility so far has been the two times I served as the instructor (holding the title of *Visiting Lecturer*) for the same functional programming course I worked as a teaching assistant during my undergraduate. I delivered twenty-one and thirty-two lectures in the summer 2020 and 2021 instances of the course, respectively, and both times was responsible for interviewing and selecting the course staff, coordinating their activities, addressing student needs, and setting the high-level direction of the course. Due to the COVID-19 pandemic, both instances were offered remotely, which presented numerous challenges. For instance, in 2020 a significant portion of the students were located in South and East Asia, requiring our US-based course staff to hold recitations and office hours at all hours of the night. I instituted a number of changes to the course to better suit the unusual circumstances, such as replacing the two midterm exams with several smaller quizzes. The challenge of remote teaching profoundly shaped me as an instructor, and led me to experiment widely with different technologies and techniques.

[1]Now known as the *Student Success Center*.

[2]Or "tutor", as its commonly known in the UK.

Teaching Philosophy

Research and experience tells us that learning is, and must be, an *active* process. If the students are not *doing*, they are not *learning*. Accordingly, my teaching is always oriented towards getting students to the point of doing exercises themselves, as soon as possible. Only once the students have an intuitive "feel" for what problem-solving is like will they be ready to appreciate higher-level abstractions which make that problem-solving work more expedient. I also favor a exercise-centric teaching approach (ideally with more frequent, smaller assessment rather than infrequent, high-stakes assessments) because it gives ample opportunity to evaluate student understanding, identify weaknesses, and give students feedback.

The easiest entry-point for getting students started on exercises are *purely mechanical procedures*. Where possible, I tried to identify problems that students could solve by rote procedure, without having to understand much at all. For instance, when I taught functional programming, one of the essential challenges was that most students had never done any functional programming, and their programming intuitions were tuned for a different style of programming, imperative programming, which would not serve them well as the course went on. In order for them to succeed, they needed to get a feel for how functional computation *worked*. So, in the early stages of the course, I assigned problems asking them to calculate out how a functional programming language would compute a given expression, according to an explicit procedure covered in lecture. These kinds of problems serve several purposes: (1) as mentioned, they get students doing things by themselves, engaging them in the *process* of learning; (2) they quickly get repetitive, stimulating the student's mind to look for broader patterns and abstractions; and (3) they provide a simple first achievement, giving confidence to students who may feel intimidated by the content.

The next level I lead my students to is *adaptation*. In my view, the most important thing to include in a lecture is a fully worked-out example of each kind of problem the students are expected to solve. It always infuriated me as a student if I was asked to solve a problem, and had no reference for what *the solution should even look like*. The worked example serves as the baseline: the first exercises should then ask the students to make minor modifications and adjustments to the solution, to adapt it for slightly-different questions. Adaptation is the first step towards general understanding—by adapting the same idea into diverse contexts, a general notion gets established in the student's mind—but it also helps focus the students on what matters. When teaching automata theory, for example, one of the most difficult topics for students is a result known as the *pumping lemma*, largely due to the lemma's complex logical structure.[3] But fully grasping this logical structure didn't need to be a prerequisite to correctly using the lemma—quite the contrary: one gets an understanding of the lemma's meaning *by* using it over and over again. So, in teaching the pumping lemma, I always proceeded by giving students some throughly-solved examples, asking them to identify which parts of the arguments were actually problem-specific and which were general boilerplate, and then getting them to do as many examples as possible on their own. In my experience, this produced students with a firm practical grasp of the lemma, and avoided tangling them up in its logic.

I am also quite mindful of the need to offer a challenge to advanced students—I have been lucky to have numerous highly-motivated and very gifted students. A well-designed exercise set will start with problems similar to what was covered in lecture, but will require increasing amounts of insight and thought. But, particularly in an exam setting, I also strive to pose students with questions requiring high-level understanding. I typically prefer to do this using short-answer questions or tricky multiple choice questions.[4] In my experience, asking extremely difficult programming or proof questions had limited probative value and had the potential to be very stressful for the students, especially students already struggling in the course.

Another axiom of my teaching is *engage students in every dimension available*. Anyone who has attended my lectures will attest that I love bright, bold colors; where possible, I try to attach meaning to the colors I use in my lecture slides, thereby creating another layer of meaning for students to

[3]The lemma is of the form *for all... there exists... such that for all... there exists... such that for all...*.

[4]Though I received some negative student feedback on this point, particularly in the 2021 functional programming course, as they felt that tricky multiple-choice/true-false questions were too "gotcha"—in the future, I plan on leaning more heavily on short answer questions to gauge high-level understanding.

engage with. I try to use the minimal amount of words as needed in these slides: anything that can be conveyed visually, should be. I also delight in incorporating vivid tactile elements to my teaching, like demonstrating *non-contractible homotopy spaces* by violently punching holes in pieces of paper, or explaining type- and syntax-checking of computer programs with the help of a satisfying 'APPROVED' stamp. I have also incorporated metaphors and themes from history, philosophy, literature, biology, chemistry, economics, graphic design, music, urban planning, card games, riddles, internet memes, sports, mythology, and much more into my teaching. I loathe the suggestion that mathematics, logic, and computer science are somehow detached from our common cultural and artistic life; I defy this suggestion, every chance I get.