Jérémie O. Lumbroso, Ph.D
Practice Assistant Professor
`lumbroso@cis.upenn.edu`

TEACHING STATEMENT (shortened)

April 15, 2023,

I love computer science deeply. My father placed me in front of a computer when I was a year and a half. Initially, the computer helped me learn how to do math, how to read. Eventually, through programming, the computer became a means of creative expression and problem solving for me. For instance, when I was 11 years old, I built software to keep track of my chores and to issue invoices to my parents. Once my programming skills grew to a certain level, I started learning about other aspects of computer science, until,, in college, I began doing theory and algorithm design. The way I learn has encouraged me to deconstruct and reconstruct concepts down to first principles, such that I am able to reexplain every part.

It is with this deep familiarity and passion for computer science that I approach my teaching. I like connecting different notions inside our curricula together, I like connecting computer science with the world, and to students' specific interests. I often feel like I am not imparting facts, but connections; that I am sharing important keys and insights with students that will help them better understand the world that is increasingly at risk of becoming "magical" by the ubiquity of complex technology[1].

Teaching philosophy. I have an excellent track record in teaching, which helped me achieve high evaluations in my later semesters (4.8/5). Additionally, I earned an Excellence in Teaching Award in 2019, and the Princeton Engineering Commendation List for Outstanding Teaching in 2020 and 2021. I try to be useful, relevant to, and supportive of students. I enjoy stimulating curiosity and imagination, creating a culture in which it is safe and encouraged to make mistakes, and where everybody's questions are an opportunity to refine our own collective understanding.

As I elaborate more in depth in my diversity and inclusion statement, I believe a large part of teaching consists of creating an *inclusive classroom*:

- I learn students' names using flash cards, so I can address them, recognize them outside of class and better remember information about them.

- I tend to avoid humor, because I think in our fast paced classes, it's easy to miscommunicate.

- I blend different kinds of assessments types (*e.g.*, in an intro course: quick quizzes, programming assignments, programming exercises, reading checkmarks, many small problems, programming exam and written exam, final project) and content (*e.g.*, in an algorithms course I will try to have students trace an algorithm but also come up with counter-examples, or debug an algorithm that is flawed, or design an algorithm to solve a problem, or provide possible and impossible outputs of an algorithm, or implement a provided algorithm), trying to give learners of all styles a variety of means to succeed.

I use *active pedagogy*, both in the context of *traditional courses* (in which knowledge is imparted by me) and *flipped courses* (in which knowledge is imparted by course materials outside of class, and I facilitate formative activities):

- I make sure that students are active participants of my classes, so that they see the value of coming, and do not need to be compelled with attendance policies.

- I center my classroom on *listening to student questions*. I often use electronic clickers and software such as Sli.do, that allows for polling students and also crowdsource questions from them. Sli.do allows

[1] David Walter, "*Scratch That: Mitchel Resnick '78 thinks coding is child's play*," Princeton Alumni Weekly, Octobre 25, 2017. https://paw.princeton.edu/article/scratch

students to propose their own questions or vote on the questions of others. In large classes, this is useful to get insights into questions shared by several students.

- I try to guide my students into making the most of their active classroom by making sure that expectations are clear. For instance, when I ask them to think about a question, I set a timer and let them know how much time they have, to ensure that they do not avoid thinking for themselves.

- When I ask a formative question, and nobody spontaneously answers, I don't call on the top student, but instead try to figure out where I lost the majority of students in my explanations.

I encourage students to think of themselves as being part of a community of learners:

- I enjoy connecting students to their peers, by reminding them regularly of the importance of networking in university, by encouraging them to work together in the classroom, in the context of group projects or explorations[2].

- I try to make all our resources—style guides, course websites, lecture notes—publicly editable through version control on GitHub, and I encourage students to contribute, so it is the responsibility of everyone to improve materials and make them clearer.

- I like to showcase the work of students to each other. When possible, I appreciate the opportunity for students to critique—both positively and negatively—each other's work. One of my research projects, codeLobby, streamlines the process of sharing and viewing each other's solution to programming assignments. (I hope similarly GPT will encourage us to introduce critical thinking activities.)

In the future I hope to move towards specification grading, for reasons I elaborate in my diversity and inclusion statement. All the courses I have taught at Princeton, were taught as a team, and I most often did not have the freedom to make significant changes to the way students were assessed. But I have tried to vary assessment methods despite this. For example, in the introductory course, I have introduced:

- "Redux exam": Since our programming exams in the introductory exam have high rates of failure, I introduced a "Redux" component, in which students have 72 hours after the exam to continue to work on it, to be graded for between 50-90% of the credit. This allows students to remember that they succeeded at this exam rather than see it as failure.

- With my tool codePost, I am able to track style deduction of the same sort across assignments. In this way, I can promise students that if they only make a programming style mistake once, I will reverse it. This focuses their attention on learning from their mistakes.

- Our quizzes are autograded instantly and randomly generated. To prevent students from randomly guessing until they get an answer, we have placed various kinds of limits: First, a fixed limit, but this encouraged students to hoard them and wait until the last minute. So to encourage early work, I allow an infinite number of try up to a certain date, and then a fixed number of try — this is to encourage students to start work early. Another approach to encourage thoughtful work, is to have a geometric delay (they can submit up to 3 answers in short succession, with a cooldown of 5 minutes if 1 attempt, 50 minutes if 2 attempts, and 500 minutes if 3 attempts).

[2] Although I am aware and have used POGIL (Process Oriented Guided Inquiry Learning), I have not had the opportunity to include it in any of our curricula yet.