Adam Meyers: Computer Science Teaching Statement

My career in Natural Language Processing began at IBM Research in 1989, while I was a graduate student in the linguistics department at New York University. I have continued working in research positions for most of the last 27 years. During this time, I have taught the equivalent of 4 years of graduate and undergraduate classes, mostly in Computer Science, but in linguistics as well. I have taught: Natural Language Processing (at both the graduate and undergraduate level, in both Computer Science and Linguistics departments); Introduction to Computers and Programming classes (undergraduate Computer Science); Syntax (graduate Linguistics); and Computer Music (a pre-college class funded by an NSF grant which resulted in a conference paper presented at ACM Creativity and Cognition 2009). My average ratings in student evaluations of my classes average slightly above 4 on a scale from 1 to 5. I have enjoyed teaching these classes and would eventually like to expand my teaching repertoire. I have a particular interest in the interdisciplinary areas of Computer Science which may fall under Data Science or Digital Humanities. As a research professor at New York University, I have mentored many students and post docs, have been on several dissertation committees and advised one dissertation. Many of these students wrote conference papers, some of which I was a co-author.

In my Natural Language Processing (NLP) classes, I teach the combination of computer science and descriptive linguistics needed to implement systems for CL applications like machine translation, information retrieval and information extraction, as well as basic technologies like natural language parsing, part of speech tagging and semantic role labeling. Homework assignments include the creation of systems for performing tasks on pre-marked test data sets, using both unsupervised and supervised machine learning, e.g., HMM part of speech tagging, Information Retrieval and Maximum Entropy based Noun Group Chunking. At the end of the semester, each student works on a final project. This includes: a project proposal due a month before the end of the semester, a conference-style talk and a conference-length paper. The final project can consist of corpus annotation with evaluation (by students or by Amazon Mechanical Turk), an implemented program with an evaluation of its performance on a task, or a paper surveying the state of the art of a subject area within NLP. I encourage students to create simple base-line systems in addition to more complex systems that include experimental ideas. This ensures that the student’s system gets results, even if it does not end up being a high-performing system. I also encourage students to work together on team projects, since many successful conference papers are team efforts. I meet with most of the students, several times in some cases, while they formulate and work on their projects. I sometimes mentor students further after the class is finished. Final projects have been developed into papers accepted at conferences. Students from my classes (graduate and undergraduate) often join my research team.

I enjoy teaching students to write programs for the first time because, for these students, it involves thinking in a new way. I argue that this way of thinking, Computational Thinking (Wing 2005, http://www.cs.cmu.edu/afs/cs/usr/wing/www/publications/Wing06.pdf), provides them with a new way of solving problems. Just as literacy changed the course of human knowledge, computational literacy of the programming variety is changing the way humans solve all types of problems. While these changes are the most obvious in the sciences (all science graduate students write programs), algorithmic based art-forms are taking root within the arts and ways to apply computational solutions to humanities subjects (e.g., computational linguistics) are
also taking root. In this way, I argue that computer science has the potential to (further) revolutionize human thought, just as writing did thousands of years ago. While, perhaps, a little dramatic, I think this is an exciting way to introduce a discipline that is very new to many college students, even those that have used computers their whole life. I suggest that a computer is more than just an appliance for surfing the web and writing papers–it is a tool that is revolutionizing the way human beings think.

I describe the process of programming and the kind of reasoning that leads to successful programming. One needs to both plan ahead (e.g., write down an algorithm) and not be afraid of trying lots of different variations when things don’t work. One needs to read and interpret error messages and use them as a guide to debugging. When carefully followed instructions don’t lead to a solution, rapid experimentation can often solve problems, and so on. I present programming as a combination of deliberate problem solving and game play. I make a point of writing code in front of the students that breaks and has error messages and I show them how to get things to work by trying lots of things in succession. I use diverse examples of programs both in class and in the homework so that it is clear that programming applies to everything: math, art, medicine, games, linguistics, spam detection, genealogy (nth cousin k times removed), etc. I attempt to instill the idea that programs can be “about” anything, just like writing. Programs can be fun, practical, experimental, scientific, artistic, etc. because programming is a means of communication and problem solving and should not be pinned down to just one type of subject or one kind of activity. Indeed, one of the things that makes computer science so revolutionary is that it encourages one to apply the scientific method and formalization very broadly, not just to scientific fields, but to any subject matter. I do my best to show how this works, taking students step by step through formalization, problem solving and programming.