Faculty Senate plenary address

[Read at the University of Pittsburgh Faculty Senate annual spring plenary session, 2014-03-19]

David J. Birnbaum

As the only representative from the humanities on the podium today, I'd like to use my allotted ten minutes to suggest that computer programming—not using programs, but creating them—is a basic skill that can be useful in research in any discipline, including any discipline in the humanities. Humanities students often think that they can't write computer programs for a variety of defeatist reasons: it's too hard, they aren't good at math, etc. In fact, computer programming is within the reach of any student, including any humanities student.

As a humanities scholar and teacher at a *research university in the digital age*, which is the theme of today's plenary, I participate in an academic culture where *computer literacy* isn't just web browsing and social media and word processing and PowerPoint. It's an academic culture where, when we see a need for computation, our ability to do our work doesn't depend on whether someone happens to have written a program that meets our needs, or on whether we can find a collaborator to do it for us. More often than not, *we can write those programs ourselves*. *It just isn't that hard.*

There are some tasks that computers do better than humans and some that humans do better than computers. Twenty-five years ago, with no background or training in computer science, I recognized that I could write my doctoral dissertation about medieval Slavic manuscripts more quickly and more accurately if I let a computer do what it was good at (not just word processing!), but there was no software that met my needs. So I learned a programming language, I wrote the programs I needed, and I finished my dissertation.

What I do in my research is find information that is hiding inside large amounts of medieval Slavic writing. I discover and analyze patterns that are difficult to discern when I just read the words with my human eyes, and I build textual interfaces for exploration and charts and graphs that make the patterns perceptible. I create digital editions that support dynamic interactive exploration, and that enable new ways of reading. Most of what I do is now called *data mining* and *graphic visualization,* but as a philologist I also recognize it as what philologists have always done: we notice things in texts and we make sense of them.

Digital humanities means many things, and the type of digital humanities that I practice and that I teach at the University of Pittsburgh is sometimes called *humanities computing*. It involves *writing original programs to conduct primary humanities research that would otherwise be impossible*. It is not a type of computing; it's a way of doing certain types of humanities research, no different from other humanities methods and methodologies. What I do when I write programs isn't computer science: it's humanities. It isn't about the computation; it's about the text. I write computer programs that let me be a more effective humanities researcher, and so do my students.

In my undergraduate "Computational methods in the humanities" course, cross-listed in eight departments, undergraduate humanities students who have never even seen a line of computer code before write, design, and implement original projects to pursue their own humanities research agendas. In one semester they learn to do what I do: they conduct the research that matters to them as humanists, letting the computer do what it's good at, and writing the programs they need to get their work done.

The idea for my course was born when I overheard some of my Russian-language students some twenty years ago lamenting that they were taking a programming course that required them to calculate a Fibonacci series. They weren't having trouble writing a program that could do that. The problem was they had no idea why they might ever want to calculate a Fibonacci series. Why couldn't they instead have learned to write programs that would be truly useful to them as humanists, that they could use to do more effective research in their home departments? Why shouldn't the starting point for what they were learning have been real humanities research questions, questions that would have been meaningful in their home departments, so that programming for them would have been a way to conduct humanities research, and not itself the primary object of study?

I'd like to suggest three ways we might think about the role of computer programming in the academic and social education of humanities students:

1. *Computer programming isn't computer science. Computer programming is more like writing.* Everyone can learn to do it, and can be given the opportunity to learn to do it in ways that are appropriate for their disciplines, just as we now require that all departments teach writing in ways that are appropriate to the discipline. You don't have to be an English composition professor to know how to write, and how to teach writing, in your discipline. *Computer programming is like writing.*

2. *Computer programming is like learning a foreign language, except that it's easier. Much easier.* In the Dietrich School we expect all students to commit the equivalent of two years to foreign-language learning, and in many languages that means four semesters of 5-credit, 5-hour-per-week courses. Those same students can learn a programming language extremely well in one 3-credit course. A medieval Islamic historian of my acquaintance spent ten years learning classical Arabic so that he could do his research. He spent just a few months learning to do things in the Python programming language that make him a more effective and successful historian. *Learning enough computer programming to do something professionally useful in the humanities doesn't take much time.*
3. *Computer programming is like cooking.* Not everyone can pull off a multi-course meal with a large guest list where the hot food is all still hot when served. But anyone can learn to cook, and being able to cook gives you dietary options you don't have otherwise. Of course there are programming tasks that require an expert understanding of computation, programming tasks that are the equivalent of catering that multi-course meal, and those tasks may well be beyond the reach of someone who hasn't been trained in computer science. But basic mastery of a programming language that is plenty good enough to support real research is within anyone's grasp. *Learning to write useful computer programs isn't like studying at a culinary academy; it's more like high-school home economics.*

What I've tried to suggest is that computer programming, like writing, is a basic skill that can be made accessible to every educated person, and that is applicable and useful in any academic discipline, including in the humanities. Computer literacy should mean more than using the Internet and using a word processor. It should mean using the computer to get things done on your own terms, where the choice of the things is determined by your research agenda, by what you want to do. The University of Pittsburgh has had *writing across the curriculum* for decades. As a faculty member in a *research university in the digital age*, the theme of today's plenary, I look forward to our now opening a dialog about ***coding*** *across the curriculum.* That dialog can bring together traditional teachers of programming, such as computer science and information science departments, and non-traditional ones, such as digital humanists. I'll put this in the context of the humanities because that's where I do my own work: digital humanities needn't be a buzzword and an esoteric specialization. Like writing, it can be an important part of a basic liberal arts education. Like language learning, it requires an investment of time, but much less time than learning a human language. And analogously to learning how to cook, it is a skill that can improve your intellectual diet.