Computational Literacy Is Literacy: Programming Professional Development (Part 1: Foundations)

There are a number of reasons why a math or science teacher or student might want to learn to program, and at the Knowles Teacher Initiative, we have been working to help teachers do just that—learn to write programs and to teach their students to write programs.

I contend that this is a worthy goal, and that learning to program is deeper than just simply the parts that tell a computer what to do. All students should learn about programming because at its heart, learning to program is learning to think computationally, which is an important cousin to problem solving, design, and scientific inquiry.

In this series of three blog posts, I will look at how we approach teaching teachers to program at the Knowles Teacher Initiative. In the first blog post, I will focus on broad notions of applicability and how teachers might use programming knowledge in their classroom. In the second post, I will describe the design principles and implementation of a professional development that taught teachers about programming and computational thinking. We will round out the series by looking at the results of the professional development program—what worked, what can be improved, and how this could be used in other contexts.

When approaching teaching teachers how to program, the first consideration is the real motivation: what are teachers going to do with programming knowledge? In my experience, this falls into three major categories (which overlap at the edges, but are still useful to think about as distinct entities). A teacher might want to learn about programming because:

> They are teaching a course that focuses on computer programming directly (such as AP Computer Science).
> They want to build tools that will help them run their classroom, like automating an attendance-taking spreadsheet.

> They want to have students engage in programming for learning, using a programming task to teach a particular piece of content.
> They want to be able to support students who want to use programming as a performance assessment task.

While the first category (programming as the content itself) is quite rightly a focus of much professional development and teacher learning, it is not the central focus of our work in programming. The majority of teachers that have asked us for help in learning about programming do not teach programming itself, but instead see programming as a means to an end in their classroom—they either want to build something themselves, or have students build something, but in the service of teaching a different STEM subject.

Sometimes, teachers do not want to teach programming as much as they want to know how to program in order to help their classroom run more smoothly. In this case, the motivation for learning programming knowledge is classroom support. For example, teachers may want to automate more mundane (but still important) elements of teacherly bookkeeping by writing spreadsheet scripts. Every teacher has a unique system that they find ideal for keeping track of their students and every teacher has to find a way to make that work within the framework of the software the school or district uses. Being able to write scripts for spreadsheets means teachers have more flexibility and power in using their system.

Turning to teachers empowering students as programmers, teachers may want to know enough about programming in different environments to provide support so that their students can use programming as part of their own learning. Programming may be used as a mechanism to embody, evaluate, or create theories and problem solutions in a classroom. For example, students learning about statistics might use a program to repeatedly make randomly generated cases, to see if their model of probabilistic outcome matched what the computer output. A physics class might tinker with a Visual Python program that simulates motion in two dimensions, adding functions that represent friction or air resistance, checking to see how the motion that the program predicts matches observations in the world. When students do this, they are engaging in programming as an analogue to problem solving or scientific inquiry. In these cases, programming is not the end in itself but rather a means to engage students in reasoning about complex and dynamical situations in ways that transcend static methods afforded by pencil and paper.

Finally, in a classroom that supports project-based learning, either with or without programming as a central piece, students may ask to create computer programs as performance assessments to represent what they have learned. A chemistry teacher I know, for example, had a student who wanted their project to involve simulating an ideal gas, with 10 simulated molecules in a box moving around and colliding with each other and the box. In this case, programming would be considered a demonstration of understanding, rather than an inquiry or problem solving tool.

The notion of computational thinking as distinct from, but related to, computer science and software engineering is relatively novel (in educational history at least), but it has gained a lot of traction in the past decade. Seymour Papert (1980) famously espoused this view in his book, *Mindstorms: Children, Computers and Powerful Ideas*, where he talked about children learning to program so that they could understand and express ideas in other content areas such as science and mathematics. More recently, a paper by Jeanette Wing (2006) identified "computational thinking" as a literacy that encompasses the fundamentally human way of thinking that is necessary to create computational artifacts (like computers and software).

Computer programming is therefore not treated at Knowles as just something programming teachers should learn about. Instead, Knowles Fellows engage with programming as an important tool, either to support their own classroom or something that their students can engage with while learning about mathematics or science content.

In the next post, I will lay out the principles that guided the creation of the Knowles Teacher Initiative's learn-to-program professional development, and then discuss the results in the third post of the series.

References

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York, NY: Basic Books.

Wing, J. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–36.

[1] Although she did not use the term literacy in her 2006 paper, the phrase "… something every human being must know to function in modern society …" captures the intent of literacy quite well.