

Courtesy of David Crusoe. Used with permission.

DavidCrusoe5

Do you think it's important for people to learn to create their own computer programs? If so, why? If not, why not?

(What a wonderful, thought-provoking question!)

What are the powerful ideas in programming?

In a sense, we need to first define what constitutes a computer program, and for what purpose the program is being created. Only then can we decide if it's important "for people to learn to create their own computer programs." In the traditional sense, a computer program is a set of commands that accomplish a specific goal through accepting, processing and outputting information. It's likely that, on average, a person won't need to build literacy in traditional programming to become successful at his or her career. It's also likely that "traditional programming" languages are quickly surpassed by newer, more powerful languages, thus rendering obsolete any "traditional" language we've learned or taught. How many of us still program in Basic? (Or, who have I caught red-handed?)

In the broad sense, thinking about programming has a tremendously practical application. To command an understanding of the "if, then, else" loop is to understand how - currently - computers process information, and so is to command the computer more effectively. It's these rules that "think out loud" GUI programming explores and explicates with and to the user. Indeed, these are simple, powerful programming languages that help users to become (as diSessa puts it) "computationally literate," rather than "computer literate." Expertise in computational literacy provides a powerful pillar for thinking with technologies.

In seeking knowledge through Google, for example, one might apply the "if-then-else" to the construction of a good search - after all, in many cases, the material retrieved is only as good as the search. Or, these syntax applications could be expanded to new domains; a learner might utilize the idea of loops, for example, to explore temperature changes in an environment. IF and ELSE might be used to direct a robot through specific actions. So perhaps it isn't programming that's important, but it's important to understand how to make a computer "think" in a specific direction.

Perhaps, Instead of asking ourselves whether it's important a learner knows how to program, we might ask ourselves: Given that it's important to understand the processes of thinking with a computer, what "powerful ideas" exist within the confines of computer programming languages to help users explore new territories? What powerful ideas are missing, and how might we create environments to simulate "thinking with powerful programming ideas?" And, perhaps most importantly, we might evaluate what are we forgetting.

We might think, for example, about a classroom environment 20 years from today. Where will it be? Will we have a sixth-grade, or will ages be mixed by capability? Which processes of thinking will be emphasized - much as "collaboration" is today - and which will be deemphasized? What skills do we need to emphasize today to work toward tomorrow's learner?

Thus, I believe we need to identify learning pursuits that computers can propel, and those "powerful programming ideas" that, through programming, will propel users in new directions.

Programming the System

I'll rant some more. This is really quite an interesting topic to think at.

What occurs to me is that, for the 21st century, we don't need to raise a flower-bed of programmers; we need instead to raise a population of hackers. Before you choke on your latte, hear me out.

The absolute volume of information will require tomorrow's thinker to be adept at pulling apart arguments - what Chris Dede calls "thriving on chaos," or "making rapid decisions based on incomplete information to resolve novel situations." To expand, tomorrow's thinker is going to need a vast array of skills to glean the logical argument beneath piles of internet dross.

Part of exploring for valid information may well include the programming of materials to gather complete information. For example, let's imagine the "model" technology-enhanced classroom. We give each student a computer, and teachers have relinquished their "Sage" status to become "partners in learning." (Ok, so I'm influenced by [ThinkQuest?](#) GPA methods)

Students collaborate to produce and construct within environments (and so are often required to tap powerful programming ideas) but, more importantly, must frame and support the problems they wish to solve. Chances are that they'll turn to Google; but how do they know the search they're conducting will yield honest results? How will they piece together the many viewpoints they'll retrieve?

Beyond simply graphically representing search results (some have tried) what if students could actually graphically *program* their query, much as we do through [StarLOGO?](#), Scratch, etc.? The programming processes we argue about today might now be used for novel, constructive purposes and -- wow! -- would be transferred out from the domain of constructing a program to the application of framing new knowledge. Perhaps this is how we'd want to raise those 21st century thinkers - to be crack programmers capable of ordering precise bits of knowledge?