Edutopia.org
The George Lucas Education Foundation

Programming: The New Literacy

Power will soon belong to those who can master a variety of expressive human-machine interactions.

by Marc Prensky
January 13, 2008

Already, various thinkers about the future have proposed a number of candidates for the designation "twenty-first-century literacy." That is, what are the key skills humans must possess in order to be considered literate? Some writers assume that the definition of *literacy* will continue to be what it always has been: "The ability to carefully read and write a contemporary spoken language." Others specify that the term will apply only to fluency in one or more of the languages spoken by the largest numbers of people, those certain to be important over the next nine decades of the century; candidates include Spanish, English, or Mandarin Chinese.

Still others expand the notion of twenty-first-century literacy beyond spoken and written language to include the panoply of skills often collected under the umbrella term *multimedia* (being able to both understand and create messages, communications, and works that include, or are constructed with, visual, aural, and haptic -- that is, physical -- elements as well as words). Some go on to find important emerging literacy in interactivity and games. And there are those who say it includes all of the above, and might include other factors as well.

I am one of these last, in that I believe fluency with multiple spoken languages will continue to be important, and that multimedia, interactivity, and other game-derived devices will be increasingly significant tools for communicating twenty-first-century thought. Nonetheless, I firmly believe that the true key literacy of the new century lies outside all these domains.

I believe the single skill that will, above all others, distinguish a literate person is programming literacy, the ability to make digital technology do whatever, within the possible one wants it to do -- to bend digital technology to one's needs, purposes, and will, just as in the present we bend words and images. Some call this skill human-machine interaction; some call it procedural literacy. Others just call it programming.

Seem strange? I'm sure it does. Today, people with highly developed skills in this area are seen as nerds. But consider that as machines become even more important components of our communication, our work, our education, our travel, our homes, and our leisure, the ability to make them do what we want will become increasingly valuable. Already, today, a former programmer in Seattle, one of these very nerds, is one of the richest people in the world.

So, in a sense, we are going to see as we progress through the twenty-first century a real revenge of the nerds, except that the new nerds will be our programmatically literate children. As programming becomes more important, it will leave the back room and become a key skill and attribute of our top intellectual and social classes, just as reading and writing did in the past. Remember, only a few centuries ago, reading and writing were confined to a small specialist class whose members we called scribes.

Do You HTML?

One might ask, "Will every educated person really have to program? Can't the people who need programming just buy it?" Possibly. Of course, with that model, we have in a sense returned to the Middle Ages or ancient Egypt, or even before. Then, if you needed to communicate your thoughts on paper, you couldn't do it yourself. You had to hire a better-educated person -- a scribe -- who knew the writing code. Then, at the other end, you needed someone to read or decode it -- unless, of course, you were "well educated," that is, you had been taught to read and write and thus had become literate.

Here's a key question: Will the need for a separate scribe tribe of programmers continue through the twenty-first century, or will the skill set of an educated person soon include programming fluency? I think that as programming becomes increasingly easy (which it will) and as the need to show rather than explain becomes important (which it will) and as people working together want to combine the results of their efforts and ideas instantaneously (which they will), educated people will, out of necessity, become programmers. Think of it: Your phone and car already require programming skills; many houses and jobs do, too. Programming will soon be how we interact with all our objects, and I believe it will be an important component of how we interact with one another as well.

Of course, there are already Luddites who think a digital machine is most elegant if it has only one button (like the Roomba [1] robot floor cleaner) and people who keep searching for a cell phone that *only* makes phone calls. (Good luck.) There is a hierarchy of levels of making machines do what you want (that is, programming them) that runs from manipulating a single on-off switch to managing menus, options, and customization to coding higher-level programming languages (Flash, HTML, scripting) and lower-level languages (C++, Java) to creating assembler or machine language.

Few people, however, remain satisfied for long with the first level -- as soon as we master that, most of us seek refinements and customization to our own needs and tastes. (The company that makes the Roomba offers a kit to turn its parts into whatever type of robot you want.)

Just about every young person programs (controls his or her own digital technology) to some extent. Many actions considered merely tasks -- setting up a universal television remote, downloading a ringtone, customizing your mobile phone or desktop -- are really programming. Doing a Web search is programming, as is using peer-to-peer or social-networking technologies, or eBay, or creating a document in Word, Excel, MySpace, or Facebook -- and toss in building your avatar in Second Life. Today's kids are such good programmers that parents who buy expensive high tech gadgets, such as camcorders or home theaters, often hand them to their children to set up (program) for them.

Today, most of this programming takes place in what I refer to as higher-level programming languages, consisting of menus and choices rather than the more flexible computer code. Of course, many people will be content with this level of programming (which still manages to baffle many "literate" adults).

But as today's kids grow up and become tomorrow's educated adults, most will go much further. At an early age, many young people learn the HTML language of Web pages and often branch out into its more powerful sister languages, such as XML and PHP. Other kids are learning programming languages like Game Maker, Flash, and Scratch [2], plus scripting language, graphics tools, and even C++, in order to build games. They learn them occasionally in school, but mostly on their own, after school, or in specialized summer camps. Why? First, because they realize it gives them the power to express themselves in the language of their own times, and second -- and perhaps even more importantly -- because they find it fun.

Want a Program? Hire a Kid

Suppose you have a need for a computer program. "Me?" you say. "Why would I have such a need?" But this possibility is not far-fetched at all. For instance, when Howard Dean ran for U.S. president a few years ago, he (or someone on his staff) had this idea: "What if we could collect contributions over the Internet?" Nobody had ever done this before, because the structure wasn't there -- the program had never been written. So he went out and found a young programmer -- an eighteen-year-old -- to write the necessary code, and within only a matter of weeks the contributions started pouring in.

Most of us have problems a computer or another digital machine could easily solve for us, if only we conceived them as programming problems: "What is my best commuting route under different weather or other conditions?" "What are my statistics in my sports (or hobbies or work), and how do they compare with those of others?" "What is the optimal configuration of my [you name it]?" "How close am I to retirement, and will I have enough money?"

We all have ideas and needs amenable to programming solutions. My guess is that the more educated and literate we are (in the tired twentieth-century sense), the more of these we have. Yet most of us "digital immigrants" -- those who came to computers and digital technology later in our lives -- never even know it. We never realize that our desire to contact certain groups of people at certain times, or to lighten the load of repetitive work (say, grading papers), or to solve certain types of puzzles (like Sudoku), are really programming problems, and quite solvable ones at that.

But some among us do realize this, and we hire young people -- often our kids, students, or employees but equally often consultants selling solutions -- to do the necessary programming for us. One result is that we nonprogrammers often get ripped off (charged a lot for something quite simple), say, by financial planners offering seemingly sophisticated tools that, were we the slightest bit "literate," we could not only write ourselves but also customize specifically to our needs.

That's not how it will be in the future. As we move further into the twenty-first century, well educated people who have needs and ideas addressable via programming will increasingly be able to recognize this fact and take matters into their own hands.

The Digital "Scribe Tribe"

Recently, programming languages "ordinary" people use have begun to emerge. Of these, one in particular -- Flash, from Adobe -- appears to be becoming a de facto standard. A great many kids in elementary school and the middle grades around the world are learning to program in Flash and are continually improving their skills as they advance through the grades. They use this tool and others like it (the Massachusetts Institute of Technology's Scratch, for example) to communicate a wide range of information and emotion -- from stories to logic to games to ideas to persuasive arguments to works of art -- all through programming. And it seems to them not nerdy but, rather, sophisticated and advanced.

The young people who do this vary greatly, of course, in the sophistication of what they can do. But sophisticated programming is becoming easier by the day. More and more premade programming objects -- code written by others that can simply be plugged in to perform certain tasks -- are available on the Internet, and are mostly free.

These databases of premade parts greatly enhance students' abilities, extend their programming and problem-solving capabilities, and shorten the time to get things done. In a sense, these bits of code are like an alphabet of programming. Recently, a friend was asked to program a "Wheel of Fortune" in Flash. Rather than taking a week to program it from scratch, he did a Web search, found something like what he wanted available free, and finished the project in an hour.

With these increasingly available and findable pieces of code, the range of what one can do and

communicate with programming can expand indefinitely. And though simpler programs such as Flash already allow a pretty good degree of sophistication, many young people, through game creation, Internet-tool creation, or other means, get into the more sophisticated programming languages of three-dimensional world building, scripting, and entirely abstract, logical programming languages such as Java and C++.

And so emerges the new scribe tribe of programmers, reaching into (and eventually becoming) the intellectual elite of the twenty-first century. Programming has already become a tool today's young people use to communicate with one another via such components as machinima (see the definition below), ringtones, emoticons, searches, photo manipulation, and games. Young people email or IM their creations to one another as we do our Word and Excel attachments, often posting them on the Internet for all to see. I bet few among us have not been the recent recipient of an emailed URL pointing us to an interesting program, a greeting card, a YouTube video, a machinima, or a game. (And, of course, Word and Excel are programming languages in themselves, with enormously sophisticated programming capabilities built in via macros and scripting.)

- **Flash:** A program that lets users create vector-based animation
- **Machinima:** "Machine cinema," in which simple tools found in video games are put to unexpected ends
- **Scratch:** An easy-to-use programming language developed by the Massachusetts Institute of Technology

As the century goes on, those who don't program -- who can't bend their increasingly sophisticated computers, machines, cars, and homes to their wills and needs -- will, I predict, be increasingly left behind. Parents and teachers often disrespect today's young people for being less than literate in the old reading-and-writing sense. But in turn, these young citizens of the future have no respect for adults who can't program a DVD player, a mobile phone, a computer, or anything else. Today's kids already see their parents and teachers as the illiterate ones. No wonder some teachers are scared to bring new technologies into the classroom -- the kids just laugh at their illiteracy.

So, as the highly literate person of 2008 might start off the day reading the *New York Times* and firing off a cleverly worded letter to the editor in response to a column, the highly literate person of 2028 may start the day ingesting the news in multiple ways with various types of stories they have programmed to be delivered in a preferred order, each at a preferred speed. And if that person feels a need to express an opinion, a simple bit of programming will allow him or her to determine all the people in the world to whom a response should go, and have it customized for each of them. Or one might program and fire off a video, an animation, or a simulation.

As the highly literate adult of today might pen a witty birthday card note for a young niece or nephew, the highly literate adult of tomorrow might program the child a game. And though today's highly literate person may enjoy a sophisticated novel or nonfiction book on a plane or train ride, tomorrow's highly literate person may prefer to change, by programming, whatever story or other media he or she is interacting with to suit individual preferences, and might then, with a little more programming, distribute those changes to the world.

And, of course, all this extends into the physical world as well thorough robotics and machine programming.

Tool Time

Tools have always been important to humans; now, intellectual tools are becoming increasingly significant. Until recently, getting an education and becoming a literate person meant learning to use the set of tools

considered essential for each field or discipline. The tools in any endeavor change and improve over time, but they generally do so quite slowly, and new tools are often invented not by ordinary people but by "geniuses." Getting an education in a field has long meant gaining mastery of its existing tools.

In this century, we will see, I think, something quite different. Using their ever more sophisticated programming skills, ordinary well-educated people will be constantly inventing new tools to solve whatever problems they have. In fact, this will be the expectation of what a literate person does. Already, in many circles (and not only scientific ones, although most are still rather geeky), one often hears someone say, "I wrote a little program to do that." And whether it's to find Manhattan addresses or to keep track of how many seconds remain until your next paycheck, a typical reaction is, "Can I get that?" to which the answer is as simple as a URL or a USB key.

It takes neither geeks nor armies of people to create useful tools via programming. A woman recently created an extremely useful program to compile and redeem her supermarket coupons. Google was created by two graduate students (Sergey Brin and Larry Page). Just one guy (Pierre Omidyar) developed the original program for eBay. Often, from these initial programming ideas come very big companies and profits. (Brin, Page, and Omidyar are all billionaires.)

But even if they don't yield huge profits, thousands -- and soon millions -- of people are beginning to create and share good programs we can all use free. Successful companies train new programmers, who then generate their own ideas and tools, in addition to the tools their companies build. Smart businesses are already searching for young people who can create these new tools -- employees who are twenty-first-century literate.

All of which brings us to an important question: If programming (the ability to control machines) is indeed the key literacy of this century, how do we, as educators, make our students literate? This problem is a particularly thorny one, because most teachers, even many of our best math and science instructors, do not possess the necessary skills, even rudimentary ones. Most of the tools (and even the concept of programming) were developed long after these teachers were born or schooled.

Can we do it by bringing working programmers into the schools? Not likely. Most of the good ones are busy programming and have no desire to teach.

The answer is not yet clear, but we can either come up with creative solutions to this real problem, or, in their absence, the kids will, as they are doing with so many things, figure out ways to teach themselves. Imagine: Literacy without (official) teachers.

Our machines are expected, thirty years from now, to be a billion times more powerful than they are today. Literacy will belong to those who can master not words, or even multimedia, but a variety of powerful, expressive human-machine interactions. If you are from the old school, you may not enjoy hearing this, but I doubt there is anything anyone can do to stop it.

Thirty years from now, will the United States be more competitive with a population that can read English at a tenth-grade level or with a population excellent at making the complex machines of that era do their bidding? The two options may be mutually exclusive, and the right choice may determine our children's place in the world's intellectual hierarchy.

Marc Prensky is the author of *Digital Game-Based Learning* and *Don't Bother Me, Mom, I'm Learning*. He is also founder and CEO of Games2train, a game-based learning company.

Source URL: http://www.edutopia.org/programming-the-new-literacy

Programming Is the New Literacy. Jan 2008. M Prensky. Prensky, M. (2008) "Programming Is the New Literacy," Edutopia, retrieved from www.edutopia.org/literacycomputer-programming.Â Since the 1960s, computer scientists and enthusiasts have paralleled computer programming to literacy, arguing it is a generalizable skill that should be more widely taught and held. Launching from that premise, this article leverages historical and social findings from literacy studies to frame computer programming as â€œcomputational literacy.â€ I argue that programming and writing have followed similar historical trajectories as material technologies and explain how they are intertwined in contemporary composition environments. Literate programming is writing out the program logic in a human language with included (separated by a primitive markup) code snippets and macros. Macros in a literate source file are simply title-like or explanatory phrases in a human language that describe human abstractions created while solving the programming problem, and hiding chunks of code or lower-level macros. These macros are similar to the algorithms in pseudocode typically used in teaching computer science.Â This is the converse of literate programming: well-documented code or documentation extracted from code follows the structure of the code, with documentation embedded in the code; while in literate programming, code is embedded in documentation, with the code following the structure of the documentation. Our modern world relies heavily on coding and programming. Perhaps it is time to make sure all children are computer literate?Â Coding, and programming, are all around us today. For this reason it might be time to make coding literacy mandatory for schools. By Christopher McFadden. Sep 07, 2019. scyther5/iStock. Coding and programming are all around us and will only get more all-pervasive in the future. For this reason, many are making the argument that learning to code, or at least have a basic understanding of it, should become a common part of a child's education. If you simply google the phrase â€œcoding new literacyâ€ (coding is the same thing as programming), you will find several viewpoints on this issue; enough to fill an entire afternoon. I am one of the proponents of this view and Iâ€™d like to offer some further opinion on why programming is an absolutely necessary skill for all students. To understand the real issue, donâ€™t think about programming, donâ€™t even worry that you may not know what that actually means. Think instead about paper.Â Although paper itself is very old, a new technology came about in mid 20th century that changed everything: digital printing (both photocopiers and printers emerged around 1950's, although printers came to wide use much later).