45

PROGRAMMING AS LITERACY

Annette Vee

Connections between writing and programming have long been commonplace in certain areas of computer science (Knuth 1992; Abelson et al. 1996). In fact, almost as soon as computers arrived on college campuses in the United States, computer programming was proposed as a burgeoning literacy akin to reading and writing. In 1961, computer scientist Alan Perlis (1962) argued that undergraduates should be taught programming just as they are taught writing in first-year composition courses. Shortly afterward, mathematicians John Kemeny and Thomas Kurtz developed the BASIC programming language, which was widely taught in schools in the 1970s and installed on many of the first home computers to hit the U.S. and U.K. in the 1980s. In U.S. schools in the 1980s, Logo promised to introduce programming on a wide scale, preparing the next generation to fight in the Cold War. Educational approaches to "computer literacy" shifted away from programming and toward usage in the 1990s, as operating systems began offering graphical interfaces and the commercial software market expanded. But, as computers have become even cheaper, more accessible, more ubiquitous, and more relevant to contemporary communication practices, the link between programming and literacy has been revived (Prensky 2008; Rushkoff 2010). Supposedly, "everyone should learn to code." What is it about programming that prompts these persistent connections to literacy and writing, and what do these connections mean?

The humanities can help to answer these questions. While computer science can tell us about what programming can do, approaches to programming from the humanities can tell us about programming's significance in developed societies: what it means for people to program computers and for programmed computers to increase their roles in our everyday lives. In particular, literacy studies has much to say about these questions because literacy research has long grappled with how people work with socially situated, technological systems of signs. As programming is tracing a trajectory similar to that taken by reading and writing in society, the theoretical tools of literacy studies aid us in understanding the history and future of the who, what, and why of programming.

With help from literacy studies, we will dive more deeply into the complex relationship between programming and writing. Programming has been rhetorically framed as writing when it is discussed as a potentially widespread ability like literacy. Its relationship to writing is under consideration when its status is debated in the law. This chapter explores the relationship between programming and writing, and what that relationship means for contemporary literacy, by introducing several influential projects that have posited programming as a literacy and then outlining some of the strange borders between writing and programming that certain

legal cases have highlighted. It concludes by pointing to several ways the humanities have already and can contribute to our understandings of computer programming as part of the environment in which humans now dwell and communicate.

The Rhetoric of Programming as a Literacy

Coding is currently a male-dominated practice. Because of historical, social, and economic barriers, women and certain minority populations are underrepresented in the population of people who can code (Ensmenger 2010). There are now countless initiatives aimed at making coding more inclusive: Code.org, Made with Code, #YesWeCode, Black Girls Code, and many more. Inertia behind these initiatives is so strong that they are often thought of as constituting a "coding for everyone" movement. They frequently select from the history and characteristics of mass literacy to frame their goals. Strategically, this framing makes sense: literacy, like programming, is a complex communicative ability facilitated by technology. Since textual literacy is much more prevalent than the ability to program, the analogies the movement makes are aspirational: "coding for everyone" is still a dream.

However, the project of mass literacy was similarly ambitious: the idea of getting *everyone* to read and write. We have naturalized these skills, but the truth is that reading and writing are complex—no more or less complex than programming. Literacy is still unevenly distributed: rates are generally lower among women worldwide, and in poor countries or among people of lower classes in richer countries. And literacy campaigns have been handmaidens to larger projects of cultural homogenization, such as "Americanization" in the early twentieth century. So there are obvious problems with holding literacy up as an ideal for the coding campaigns. Yet literacy rates exceed 95 percent in all industrialized countries, and the worldwide literacy rate is around 85 percent. Almost 90 percent of men and over 80 percent of women in the world can read and write by the time they are fifteen (UNESCO Institute for Statistics 2015). The dream of mass literacy has largely come true, although people's opinions differ on what it looks like and how to measure or teach it. Still, governments and citizens almost universally support literacy as a concept and as a moral good. While computer science educators can debate how we might teach "coding for everyone," the humanities are situated to ask different questions, including whether the "coding for everyone" movement *should* follow the path of mass literacy.

To help us consider this question, we can look to the historical treatment of programming as a literacy (Vee 2013). The rhetoric of programming as a literacy begins in 1961 with an address by computer scientist, Alan Perlis, at a conference at MIT. Perlis (1962) argued that all undergraduates should be taught programming just as they are taught writing in first-year composition courses. First-year composition courses were (and largely still are) seen as service courses to the rest of the university, venues for imparting the written communication skills that are essential to success in college and beyond. Perlis's argument that programming was similarly essential to undergraduate education implies that it would become foundational to many different disciplines and to civic life more generally. His vision is particularly striking given the state of computers at the time: when he gave his address, only a few college campuses had mainframe computers. But computers were increasingly important to large-scale business and government, including defense. His emphasis on broad undergraduate education in programming suggested that future leaders of America should know something about how to write for these machines.

Perlis's vision was at least partially realized with the BASIC programming language, designed at Dartmouth during the early 1960s by John Kemeny and Thomas Kurtz. Like

Perlis, Kemeny and Kurtz (1968) saw the computer as universally relevant. They designed BASIC to be accessible to undergraduates in the liberal arts as well as those in engineering or hard sciences. Kemeny (1983) in particular was concerned that future leaders trained at Dartmouth should know something about computation, given its increasingly important role in national infrastructure. BASIC was successful. By the 1968 fall term, 80 percent of Dartmouth undergraduates, plus several hundred faculty, had learned how to write computer programs (Kemeny & Kurtz 1968). In part because Kemeny and Kurtz made BASIC and their time-sharing system free, the programming language spread across college campuses during the 1960s and has enjoyed a long life afterward as well.

During the 1960s, efforts to teach programming broadly were focused on undergraduates, since computers could be found only in government offices, large corporate centers, and some college campuses. But, as the technology and culture of computing spilled out of college campuses in the 1970s and West toward California, the impetus to promote programming to the masses seems to have taken the same direction. West coast programming initiatives were imbued with post-60s San Francisco area politics: hobbyists and hackers thrived, typified by the Homebrew Computer Club, the People's Computer Company, and Ted Nelson's manifesto *Computer Lib/Dream Machines*. Nelson argued, "If you are interested in democracy and its future, you'd better understand computers" (1987 [1974]: 5). In the 1970s, computer programming—usually in BASIC—was widely taught in high school after-school programs and accelerated math classes.

By the mid-1980s, computers became cheaper and easier to use as well as more common in middle-class American households, businesses, and schools: U.S. computer ownership almost doubled from 1984 to 1989, from 8.2 percent to 15 percent, according to the U.S. Census Bureau. Affordable and accessible personal computers, such as the Commodore 64 and Apple II, meant that, for the first time, computers became available to children. The idea of computers as tools for children owes much to educational research done in the 1970s, especially by Seymour Papert and Alan Kay. Papert's team developed the Logo programming language in the U.K. and piloted it in classrooms during the late '60s and '70s. A student of Jean Piaget, Papert (1980) argued that programming could scaffold young children to work with concepts often considered too complex for their age. At Xerox PARC in the 1970s, Alan Kay led a team of researchers to prototype the Dynabook laptop as well as the programming language Smalltalk, both of which were designed to make computing and programming more accessible to children (Kay 1993; Kay & Goldberg 2003 [1977]). This research preceded the home computer revolution and provided a frame that allowed computer programming and computer usage (largely still fused at this point) to be considered general rather than specialized skills.

Efforts to widen access to computers and programming in America were focused on K-12 education in the 1980s, because of not only the efforts of people like Papert and Kay, but also the heightening of the Cold War. American schools adopted programming in Logo or BASIC as part of their curriculum, funded by Cold War initiatives for math, science, and technology preparation. These initiatives imagined the front lines on the grounds of both nuclear and computational technologies, and so children's literacy with programming was seen as critical to national defense and health during the 1980s. The term "literacy" becomes more common in the rhetoric surrounding mass programming education at this point, perhaps because literacy is often thought of as a primary goal of elementary education and, since the nineteenth-century mass education movement, children have often been the site where literacy is measured.

While the "computer literacy" movement went mainstream in the 1980s, a shift in computer technology that followed the wave of home computers began to fundamentally

change what it meant. The commercial software era severed computer programming from computer usage. Because people no longer needed to know how to program in order to use computers, the idea of computer literacy came to mean knowledge of file structures, saving work to disks, and menu operations (diSessa 2000). This utilitarian and skills-based idea of computer literacy stripped it of the optimism associated with earlier initiatives. Moreover, Papert's Logo pilot program did not scale well: uneven resources and support for teachers deflated the program's efficacy. Critics pointed to a lack of promised gains in children's cognitive abilities (Pea & Kurland 1984); and, despite Papert's open disappointment in the implementation and lack of follow-through, most programming initiatives in elementary schools were eventually phased out.

In the 1990s, the energy behind the mass programming movement shifted from elementary schools to the new World Wide Web, which once again widened access to programming. The architect of the web, Tim Berners-Lee, insisted on technical and organizational protocols that would enable it to be accessible and programmable by everyone (Berners-Lee & Fischetti 1999). For their introduction to programming, many people today credit HTML, the simple markup language on which the web is built. HTML does not have the computational capabilities of BASIC. However, BASIC inspired it, and it shares BASIC's accessibility and ubiquity. As the authors of *10 PRINT* (a collaborative book based on a BASIC maze program widely circulated during the 1980s) write, "HTML . . . copied BASIC's template of simplicity, similarity to natural language, device independence, and transparency to become many users' first introduction to manipulating code" (Montfort et al. 2012: 192). The possibilities of HTML and its ease of use, plus the fact that it led users to other more extensive languages, such as JavaScript, Perl, and PHP, enacted that gentle novice-to-expert climb that Kemeny and Kurtz sought for BASIC. However, even with most web browsers' capability to show the source code for any webpage, people are not automatically exposed to code now in the same way that they were on their Commodore 64s in the 1980s. Many mass programming advocates have pointed to this lack of exposure as a problem (why the lucky stiff 2003). While people enjoy What-You-See-Is-What-You-Get (WYSIWYG) interfaces, any interface makes assumptions about its uses and users. People learn to use menus with these assumptions embedded, rather than thinking about or creating alternative kinds of software.

The 2000s took the "coding for everyone" movement into diverse online communities with little connection to formal computer science and institutions. In his 1999 DARPA grant application, Guido van Rossum, the designer of the Python programming language, tapped into the positive cultural associations of literacy to secure funding for his dream of coding for everyone:

> We compare mass ability to read and write software with mass literacy, and predict equally pervasive changes to society. Hardware is now sufficiently fast and cheap to make mass computer education possible: the next big change will happen when most computer users have the knowledge and power to create and modify software.
>
> (van Rossum 1999)

His initiative was only funded for a year, but it signaled a burgeoning online education movement where people could learn programming by downloading copies of language compilers and developing environments, asking questions on forums such as Slashdot and Usenet, contributing to open source projects and getting feedback on SourceForge, and watching videos or reading blogs posted by people motivated to teach programming for fun, exposure, or profit. Resources for learning programming outside of computer science had never been

greater. The growth of the web allowed for ready circulation of open source programming languages, such as JavaScript, Python, Perl, PHP, and Ruby, some of which augment the computational possibilities of HTML and echo BASIC's appeal to novices.

In the 2010s, the decentralized culture of how-to videos and forums online have become consolidated by programming-promoting organizations such as Code.org, Khan Academy, and Codecademy, which feature video lessons, e-books, interactive online code-checking, and a wealth of other resources helpful to anyone wanting to learn programming. These groups echo more general interest in mass online education, typified by Lynda.com, TED Talks, and free lectures from universities through edX, as well as massively open online courses (MOOCs) through for-profit entities like Coursera. Motivations for these groups vary from profit to social justice, although all reflect an emphasis on education outside of formal programs and institutions and position students as "neoliberal" workers "both empowered and wanting (e.g., always in need of training)" (Chun 2011: 59).

The rhetoric of the current "coding for everyone" movement echoes the "empowerment" of earlier efforts. In popular commentary, Douglas Rushkoff says that learning programming gives people "access to the control panel of civilization" (2010: 1), and Marc Prensky argues "[a]s 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" (2008). Code.org, a nonprofit launched in 2013 and supported by Mark Zuckerberg and Bill Gates, showcases on their website a litany of quotes from educators, technologists, and public figures claiming that learning to code is an issue of "civil rights," the "4th literacy," and a way to "[c]ontrol your destiny, help your family, your community, and your country." Black Girls Code reflects literacy ideologies of wider access and "mastery" in their mission statement: "By promoting classes and programs we hope to grow the number of women of color working in technology and give underprivileged girls a chance to become the masters of their technological worlds" ("About us" 2014). Black Girls Code and Code.org are very different organizations with different funding structures and goals, making it especially interesting that they both draw on the tropes of literacy empowerment to promote programming education to a wider population. While some of the other initiatives discussed here do not use the term "literacy," they often draw on a similar profile of empowerment, social justice, citizenship, and productivity.

As this brief history of the "coding for everyone" movement demonstrates, programming has long been touted for its artistic possibilities as well as its utility for workers, businesses, and government applications. These attributes are also often ascribed to literacy and have been leveraged in mass literacy campaigns connected to democracy, defense, and mass schooling. References to literacy in the promotion of programming can help to convey individual as well as collective goals and responsibilities because literacy is an individual skill, but one which gains meaning and value in social contexts. Literacies purportedly help workers and boost national productivity. It is for these reasons that Code.org could recruit both U.S. President Barack Obama and a political rival, House Majority Leader Eric Cantor, in 2013 to make video statements in support of their project: programming, like literacy, is good for the country.

How Is Programming Like Writing?

The promotion of programming on a wide scale borrows from the ideas of mass literacy movements. Underneath this rhetoric, the parallels between programming and writing are quite apt. Written text and code are both symbolic systems operating through an inscribed

language and social contexts. Symbolic systems such as text and code can be distinguished from other important technologies, such as carpentry, because they are "media machines" that "process texts, images, and sound" (Poster 2011: x). Because of the ways that code can process infinite forms of information, programming is the language that transforms computers into what Alan Turing called the "universal machine."

One way for us to explore how programming is like writing is to look at the ways programming is treated like writing in United States law. (While the legal history I provide here is specific to the U.S., international copyright agreements such as the Berne Convention mean that the effects of U.S. law are similar across most developed countries.) In 1980, the U.S. Congress amended the 1976 Copyright Act to define computer code as a "literary work" and a "form of writing" when it was seeking a way to protect the intellectual property of the growing software industry. This amendment and its surrounding debate cemented computer programming as a form of creative expression. By 1994, prominent legal scholars noted, "virtually all nations have recognized the textual character of program code in deciding to use copyright law to protect it" (Samuelson et al. 1994). Somewhat controversially, copyright protection extends not only to source code (the code generally written by and for programmers to read) but also object code (the code that the computer reads). In other words, programming produces copyrightable "writing" addressed to people as well as computers (Vee 2012).

If programming produces "literary" writing, then it can be protected by First Amendment or "free speech" rights in the United States. Activities protected as "speech" under the First Amendment include oral communication, textual writing, methods of dress, gestures, visual art, and computer code. "Speech" is unprotected when it verges on conduct, such as yelling "fire" in a crowded theater. Since code is both conduct and writing—it *does* things as well as *says* things—it occupies a liminal space in these laws. Testing this boundary in U.S. courts was a series of cases involving cryptographic code written by a PhD student, Daniel Bernstein, that the U.S. State Department argued should be regulated as a "weapon." In its decision, the court rejected the State Department's argument that speech that performed a function was not entitled to protection: recipes and sheet music were clearly protected speech, yet they enabled people to do something. No form of communication can be divorced from its function, argued the court; and so computer programming, while it does *do* things, also *says* things: code is, therefore, a protected form of writing or speech.

The *Bernstein* case tested the lines between code's status as creative expression and a potential weapon. A new case involving plans for a 3-D printed gun pushes this debate further. "Wiki weapon" designer, Defense Distributed, uploaded plans to the internet that would allow a 3-D printer to produce a working handgun. When the State Department pursued them with the same weapons law used against Bernstein, they argued that their First Amendment rights had been violated. Their founder, Cody Wilson, asks: "If code is speech, the constitutional contradictions are evident . . . So what if this code is a gun?" (Greenberg 2015). Here, as in *Bernstein*, the question about how programming is related to writing is more than just academic. If sticks and stones rather than words break bones, then what does an executable blueprint for a gun do?

Writing has always been a potentially dangerous activity, Deborah Brandt (2014) argues, and it has consequently enjoyed fewer legal protections than reading. Is the situation different for programming? Because code does things, it is currently regulated by patent law; because it is writing, it is also governed by copyright law; and because it is creative expression, it is protected by law against censorship. If we think of programming as literacy, what kinds of legal protections *should* pertain to the writing or reading of code?

Conclusion

As these border cases demonstrate, programming is not separate from writing or any other form of human communication. Algorithms interoperate with our lives, including our music and movie tastes and even the heating of our homes (e.g., Pandora, Netflix, and the Nest thermostat), and necessarily combine both computational and human procedures to do their work. Self-taught algorithms take in and respond to analog input, leading to levels of complexity neither computers nor humans can fully understand alone (Hayles 2005; Aneesh 2006). Moreover, computational and programmable devices are now joining a panoply of other read-write tools to form a new matrix of social practices and technologies that constitute literacy. Programming plays a supportive role in traditional writing through word-processing programs, and it facilitates new forms of written communication such as tweets, texts, Facebook posts, emails, and instant messages. It also remaps communication when it contributes to what A. Aneesh (2006) calls "algocracy" (rule by algorithm) or acts as a blueprint for printed objects such as guns. Programming is a kind of writing, *and* it is bound up with traditional writing, *and* it exceeds historically assumed boundaries of writing. Popular discourse about "programming as a new literacy" as well as legal cases debating the nature of code reflect the complicated relationship between writing and programming.

Writing and programming are both/together inextricable and important forms of composition and communication. If *literacy* implies an ability to wield the dominant tools of communication in a society, then in contemporary networked societies it includes some degree of facility with computer programming. But literacy to what degree? For whom, and why? How might people go about learning this new kind of literacy? And how should the activities of literacy be protected through our social and legal institutions? These value-laden questions have been considered by literacy studies in terms of textual writing and reading. In terms of programming, they are currently being considered by fields such as digital rhetoric, digital humanities, and software studies, which are developing theoretical tools to consider the material and social affordances of digital technologies in human lives. While computer scientists have considered the unique challenges in teaching novice programmers (Soloway & Spohrer 1988) and what education in computational literacy might look like (Wing 2006; Guzdial 2008), approaches to these questions from the humanities can help us consider questions of politics and identity embedded in *who* learns computational literacy and *why*.

Further Reading

Brandt, D. (2014) *The Rise of Writing: Redefining Mass Literacy*. Cambridge, U.K.: Cambridge University Press.

Chun, W. H. K. (2011) *Programmed Visions: Software and Memory*, Cambridge, MA: MIT Press.

diSessa, A. (2000) *Changing Minds: Computers, Learning and Literacy*, Cambridge, MA: MIT Press.

Guzdial, M. (2015) *Learner-Centered Design of Computing Education: Research on Computing for Everyone*, San Rafael, CA: Morgan & Claypool Publishers.

References

Abelson, H., G. Sussman, and J. Sussman (1996) *Structure and Interpretation of Computer Programs*, 2nd edn. Cambridge, MA: MIT Press.

Aneesh, A. (2006) *Virtual Migration: The Programming of Globalization*, Durham, NC: Duke University Press.

Berners-Lee, T. and M. Fischetti (1999) *Weaving the Web*, New York, NY: HarperBusiness.

Black Girls Code (2014) "About Us," retrieved from www.blackgirlscode.com/what-we-do.html.

Bolter, J. D. (1991) *Writing Space: Computers, Hypertext, and the History of Writing*, 1st edn. Hillsdale, NJ: Lawrence Erlbaum.

Brandt, D. (2001) *Literacy in American Lives*, Cambridge, U.K.: Cambridge University Press.

Brandt, D. (2014) *The Rise of Writing: Redefining Mass Literacy*, Cambridge, U.K.: Cambridge University Press.
Chun, W. H. K. (2011) *Programmed Visions: Software and Memory*, Cambridge, MA: MIT Press.
Code.org. (2013) "Leaders and Trend-setters All Agree on One Thing." retrieved from www.code.org/quotes.
diSessa, A. (2000) *Changing Minds: Computers, Learning and Literacy*, Cambridge, MA: MIT Press.
Ensmenger, N. (2010) *The Computer Boys Take Over*, Cambridge, MA: MIT Press.
Greenberg, A. (2015) "3D Printed Gun Lawsuit Starts the War Between Arms Control and Free Speech," *Wired*, retrieved from www.wired.com/2015/05/3-d-printed-gun-lawsuit-starts-war-arms-control-free-speech/.
Guzdial, M. (2008) "Education: Paving the Way for Computational Thinking," *Communications of the ACM* 51(8), 25–27.
Hayles, N. K. (2005) *My Mother Was a Computer*, Chicago: University of Chicago Press.
Kay, A. (1993) "The Early History of Smalltalk," *ACM SIGPLAN Notices* 28(3), 69–95.
Kay, A. and A. Goldberg (2003 [1977]) "Personal Dynamic Media," in N. Wardrip-Fruin and N. Montfort (eds.) *The New Media Reader*, Cambridge, MA: MIT Press, pp. 393–404.
Kemeny, J. (1983) "The Case for Computer Literacy," *Daedalus* 112(2), 211–30.
Kemeny, J. and T. Kurtz (1968) "Dartmouth Time-Sharing," *Science* 162(3850), 223–28.
Knuth, D. (1992) *Literate Programming*, CSLI Lecture Notes, United States: Center for the Study of Language and Information.
Montfort, N., P. Baudoin, J. Bell, I. Bogost, J. Douglass, M. C. Marino, M. Mateas, C. Reas, M. Sample, and N. Vawter (2012) *10 PRINT CHR$(205.5+RND(1)); : GOTO 10*, Cambridge, MA: MIT Press.
Nelson, T. H. (1987) *Computer Lib / Dream Machines*, Redmond, WA: Microsoft.
Ong, W. (1982) *Orality and Literacy*, London: Routledge.
Papert, S. (1980) *Mindstorms: Children, Computers, and Powerful Ideas*, New York, NY: Basic Books.
Pea, R. D. and D. M. Kurland (1984) "On the Cognitive Effects of Learning Programming," *New Ideas in Psychology* 2(2), 137–68.
Perlis, A. (1962) "The Computer and the University," in M. Greenberger (ed.) *Computers and the World of the Future*, Cambridge, MA: MIT Press.
Poster, M. (2011) "Introduction," in Flusser V. *Does Writing Have a Future?* Minneapolis, MN: University of Minnesota Press.
Prensky, M. (2008) "Programming Is the New Literacy," *Edutopia*, retrieved from www.edutopia.org/literacy-computer-programming.
Rushkoff, D. (2010) *Program or Be Programmed: Ten Commands for a Digital Age*, OR Books. Kindle file.
Samuelson, P., R. Davis, M. D. Kapor, and J. H. Reichman (1994) "A Manifesto Concerning the Legal Protection of Computer Programs," *Columbia Law Review* 94(8), 2308–2431.
Soloway, E. and J. C. Spohrer (eds.) (1988) *Studying the Novice Programmer*, London: Psychology Press.
UNESCO Institute for Statistics (2015) "Adult and Youth Literacy," *United Nations Educational, Scientific and Cultural Organization*, UIS Fact Sheet 32, retrieved from www.uis.unesco.org/literacy/Documents/fs32–2015-literacy.pdf.
U.S. Census Bureau (2009) *Households with a Computer and Internet Use 1984–2009*, retrieved from www.census.gov/hhes/computer/.
van Rossum, G. (1999) Computer Programming for Everybody (Revised Proposal), Reston, VA: *Corporation for National Research Initiatives*, retrieved from www.python.org/doc/essays/cp4e.
Vee, A. (2012) "Text, Speech, Machine: Metaphors for Computer Code in the Law," *Computational Culture* 2, retrieved from computationalculture.net/article/text-speech-machine-metaphors-for-computer-code-in-the-law.
Vee, A. (2013) "Understanding Computer Programming as a Literacy," *Literacy in Composition Studies* 1(2), 42–64.
why the lucky stiff (2003) "The Little Coder's Predicament," retrieved from viewsourcecode.org/why/hacking/theLittleCodersPredicament.html.
Wing, J. (2006) "Computational Thinking," *Communications of the ACM* 49(3), 33–35.