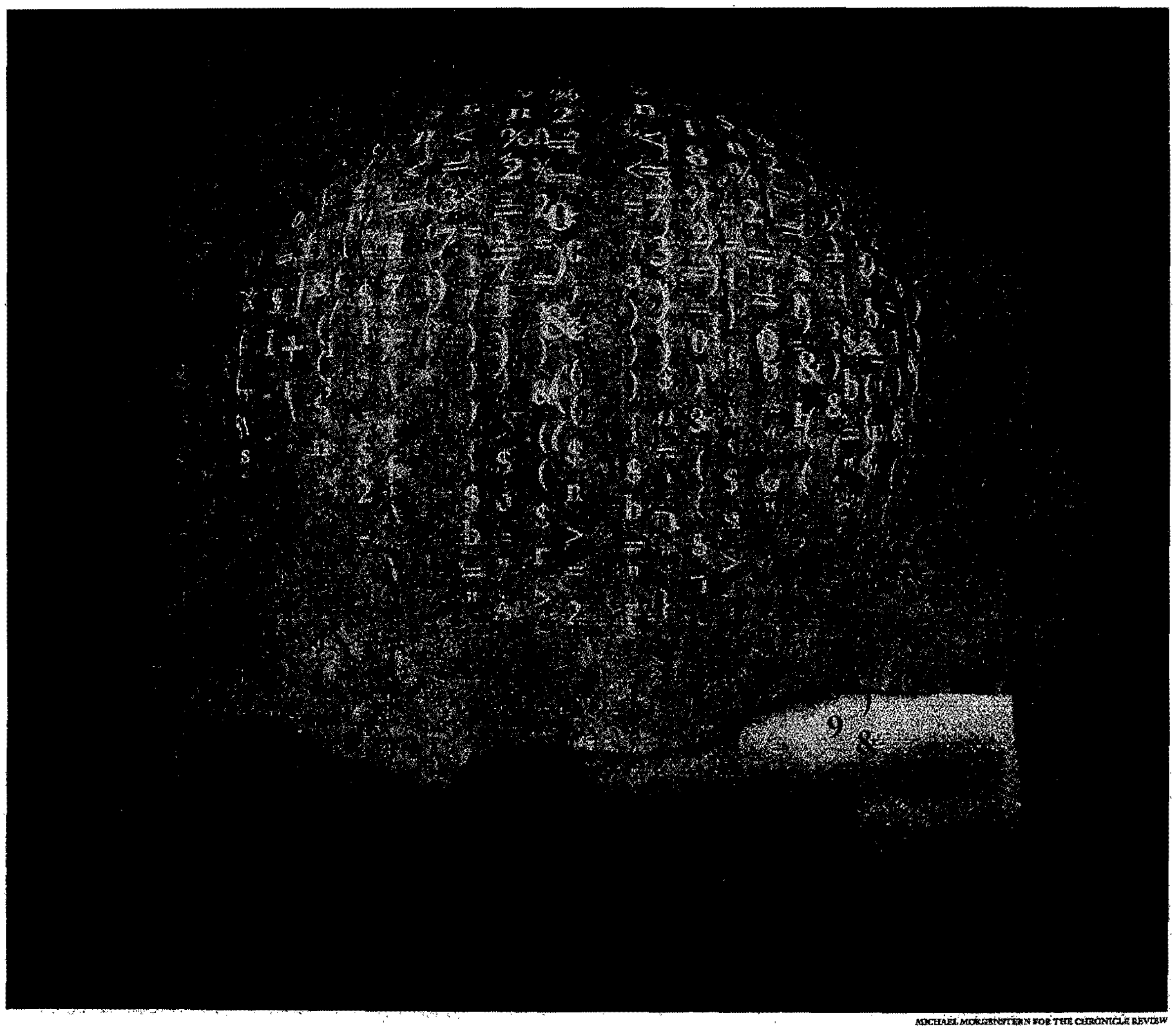
MICHAEL MORGENSTERN FOR THE CHRONICLE REVIEW

Hello Worlds

Why humanities students should learn to program

By MATTHEW KIRSCHENBAUM

THE FIRST PROGRAM most people learn to write in any computer language is called Hello World. Its sole function is to display those two words on the screen. But the act of writing and then running Hello World can raise some intriguing questions: Who, or what, exactly, is saying hello to the world? The original author of the program? The neophyte who just transcribed it on a computer? The computer itself? All of these somehow together? Whose "world" is being greeted? The world around us, or the virtual world inside the machine? Is anyone (or anything) expected to return the salutation? Hello World, whose syntax varies from one computer language to another, is a postmodern cultural artifact, and to me such questions are irresistible.

Proficiency in a computer language can fulfill many of the same functions, like heightened critical awareness, as knowledge of a foreign language.

As a computer-savvy English professor, I'm often asked how I came by my knowledge. Part of the answer is about generation and privilege: I grew up in a household that had both books and keyboards. When personal computers hit the consumer market in the early 1980s my parents brought home an Apple IIe. An afternoon moving back and forth from my novel on the couch to a program I was tinkering with on the Apple's monochrome screen didn't seem odd or discordant at all. Trampling over C.P. Snow's two-cultures divide was something I and other kids like me did every day after school.

Once I became an undergraduate English major, I took computer-programming courses to fulfill my institution's general-education requirement. I learned BASIC and Pascal. But I hated the classes. At least at the time, the pedagogy was purely vocational. Apparently I was being prepared for a life where I might own a hardware store in Schenectady (say) and would need to write my own programs to manage the store's inventory. The assignments dictated this or that functionality and constraint that needed to be implemented: I might have to sort the inventory by product name or manufacturer, or have the program generate a warning message whenever items were low in stock.

On the one hand, I can see now that I was being taught fundamentals: variables, arrays, sorts, conditionals, operators, and the like. Absent, on the other hand, was any sense of why anyone would want to learn programming; why programming was a unique and startling way of looking at the world; why it was, in fact, a kind of world-making, requiring one to specify the behaviors of an object or a system from the ground up; why and how such an activity was connected to the long traditions of humanistic thought I encountered in the classes devoted to my major, reading Leibnitz for example or (better) Jane Austen, surely one of our ultimate system builders and world-makers. What was lacking, in other words, was any kind of historical and intellectual context for why "bubble sorts" and "do-while loops" mattered.

Many of us in the humanities think our colleagues across the campus in the computer-science department spend most of their time debugging software. This is no more true than the notion that English professors spend most of their time correcting people's grammar and spelling. More significantly, many of us in the humanities miss the extent to which programming is a creative and generative activity. Many different ways exist to do even something as uninspiring as writing software to manage a retail inventory. Programming is about choices and constraints, and about how you choose to model some select slice of the world around you in the formal environment of a computer. This idea of modeling is vital, and what I think was missing from those early undergraduate courses I took. If only someone had told me I wasn't learning to manage a hardware store, I was learning to build models.

THOSE MODELS are made of words and symbols which, by their nature, can be intimidating. To the uninitiated, a computer program is inscrutable, illegible. The irony, in fact, is that Pascal or Java or C++ are also illegible to the machine. These languages are precisely meant to be human readable. In order to be readable by a computer, they first have to be translated ("compiled") into an even more abstract expression known as machine code. The formalism and constraint inherent in these so-called high-level languages are there to ensure that they can be compiled and expressed accurately as machine code. Donald Knuth, perhaps our most famous living computer scientist, famously embraced the idea of "legible code" out of the conviction that the most important audience for any computer program is a human being (because the code will be continuously worked over and shared by numerous different programmers over the course of its existence).

Many programmers talk openly of the aesthetics of code, using terms like beautiful or artful in the same way that a grandmaster might describe a game of chess (another formal activity par excellence). There is even a field known as software forensics, in which quirks and tics in a suspect program's language (the source code of a virus, say) are exploited to trace them to an individual human author, much like forensic linguists exploit stylistic features to attribute anonymous texts.

The literary avant-garde has discovered computer languages, with so-called code work emerging as a new poetic genre. Code work blends functional computer code with creative composition. Perl poems are popular; they consist of valid source code that has both compelling lyrical content and is functionally executable as a working program. Chef is a playful experimental language whose syntax yields both working programs and recipes for edible dishes (not as perverse as it might sound, since recipes resemble algorithms). Novelists have also expanded their range to include programming languages; Ellen Ullman, for example, includes chunks of raw source code in the pages of her novel *The Bug*. The effect is perhaps reminiscent of those readers who made it through Umberto Eco's *The Name of the Rose* only to find that a key line of the book is in Latin. What is surprising to many of Ullman's readers is just how much of the code they can make some sense of, even if they are not (like Berta, her novel's protagonist) code savvy. Clearly the distinction between what's on the screen (or page) and what lies beneath is beginning to disappear, as computer languages seep into the visible, legible spaces in which we read.

I BELIEVE SUCH TRENDS will eventually affect the minutiae of academic policy. The English department where I teach, like most which offer the doctorate, requires students to demonstrate proficiency in at least one foreign language. Should a graduate student be allowed to substitute demonstrated proficiency in a computer-programming language instead? Such questions have recently arisen in my department and elsewhere; in my own case, almost a decade ago, I was granted permission to use the computer language Perl in lieu of proficiency in the second of two languages that my department required for the Ph.D. I successfully made the case that given my interest in the digital humanities, this was far more practical than revisiting my high-school Spanish. Was that an equitable substitution, or was I comparing apples and oranges?

For now these questions should be resolved on a case-by-case basis. Knowledge of a foreign language is desirable so that a scholar does not have to rely exclusively on existing translations and so that the accuracy of others' translations can be scrutinized. One also learns something about the idiosyncrasies of the English language in the process. A computer language will not replace the comparativist's need to know Spanish or French or German, or the budding medievalist's command of Latin and Greek. But what about the student of contemporary literature interested in, say, electronic poetry or the art of the novel in the

Continued on Following Page

Where Computer Science and Cultural Studies Collide

ACADEMIC STUDY of new media is increasingly spawning more specialized inquiry: Game studies, software studies, critical-code studies, even platform studies are all buzzwords in the field. One high-profile meeting last year at the University of California at San Diego, hosted by Lev Manovich and Noah Wardrip-Fruin, brought together an international array of scholars, artists, and technologists, including many of the names mentioned in the accompanying article, to discuss "the meaning of studying software cultures, and the direction and goals of software studies as an emerging movement" (http://workshop.softwarestudies.com). The centerpiece of the meeting, a public "pecha kutcha" session, featured several dozen seven-minute speed talks with titles like "Expressive Processing," "Unmanned Systems as Assemblages," "The Time of Codework," and "Software Values."

Most users have no more knowledge of what their computer or code is actually doing than most automobile owners have of their carburetor or catalytic converter. Nor is any such knowledge necessarily needed. But for academics, driven by an increasing emphasis on the materiality of new media—that is, the social, cultural, and economic factors driving technical innovation, essentially the inverse of an old-school technological determinism that posited technology as the governing force in human activity—no hardware component is too exotic, no acronym too esoteric to escape critical notice. Put another way, software studies and its kin are the collision of computer science and cultural studies.

No hardware is too exotic, no acronym too esoteric to escape critical notice.

That collision happens on a daily basis at one of the busiest crossroads in academic new media, a group blog called Grand Text Auto (http://grandtextauto.org). Its authors, Mary Flanagan, Michael Mateas, Nick Montfort, Scott Rettberg, Andrew Stern, and Noah Wardrip-Fruin, are all hands-on developers and creators of new media like interactive fiction and games, as well as critics and theorists. In the five and a half years since it's been online, Grand Text Auto has become the single must-read blog for the field.

The MIT Press, which has an extensive list in new media, last year published *Software Studies: A Lexicon*, edited by Matthew Fuller and featuring contributions by a number of key critics and thinkers. The MIT Press is also launching a series in platform studies, edited by Nick Montfort and Ian Bogost (http://platformstudies.com). According to Montfort and Bogost, "Platform studies investigates the relationships between the hardware and software design of computing systems and the creative works produced on those systems." They have written the inaugural volume, a study of the Atari 2600 Video Computer System, to be published this year. My own *Mechanisms: New Media and the Forensic Imagination* (MIT Press, 2008) is likewise in this vein.

Critical Code Studies is the mantle of another group blog, founded by Jeremy Douglass and Mark Marino, whose mission is to "promote the close reading of software within socio-historical contexts" (http://criticalcodestudies.com/wordpress). Critical-code studies have also been the focus of sessions at the Modern Language Association and the Society for Literature, Science, and the Arts. In an essay in the electronic book review, Marino writes that critical-code studies "holds that lines of code are not value-neutral and can be analyzed using the theoretical approaches applied to other semiotic systems" and that it "follows the work of critical legal studies, in that its practitioners apply critical theory to a functional document (legal document or computer program) to explicate meaning in excess of the document's functionality, critiquing more than merely aesthetics and efficiency" (http://www.electronicbookreview.com/thread/electropoetics/codology).

Rita Raley's book on new media and code, to be published later this year by the University of Minnesota Press, promises to further extend these inquiries.

Games studies is the most venerable of these various offshoots of new-media studies, and also the one whose relationship to the larger discipline is most fraught. Almost anyone familiar with academic discussion about computer games has probably encountered vestiges of a once-vociferous game-playing versus storytelling debate, which largely defined the field for several years following the 2001 launch of its foundational journal, *Game Studies*, edited by Espen Aarseth (http://gamestudies.org). Today game studies has been invigorated by a wave of writing, including the *First Person* and *Second Person* compilations edited by Pat Harrigan and Wardrip-Fruin (*Third Person* is forthcoming later this year); *Digital Culture, Play, and Identity: A World of Warcraft Reader* (edited by Hilde G. Corneliussen and Jill Walker Rettberg); and Bogost's pathbreaking study *Persuasive Games* (all from MIT); as well as Alex Galloway's books *Gaming: Essays on Algorithmic Culture* and *The Exploit: A Theory of Networks*, with Eugene Thacker (both from Minnesota), and McKenzie Wark's *Gamer Theory* (from Harvard University Press).

Meanwhile, Stephen Ramsay's work on "algorithmic criticism," which explores the role of procedure in literary criticism and applied digital humanities, represents still another important area of activity and serves to complement the kind of projects described above. Ramsay's book is forthcoming from the University of Illinois Press, which has begun a series in digital humanities edited by Susan Schreibman and Ray Siemens. ■

—M. K.

Continued From Preceding Page

information age? Or the student interested in computer-assisted text analysis, who may need to create specialized programs that don't yet exist? For these students, I believe proficiency in a computer language can fulfill many of the same functions—accessibility, self-reliance, heightened critical awareness—as knowledge of a traditional foreign language.

WHEN I TEACH COURSES on new media and electronic literature, I'm not interested in turning my students into professional code monkeys. They can go elsewhere for that if that's what they want. Instead, I like to start by making snowballs with them. More precisely, we use computers to make models of snowballs, in an object-oriented programming environment called a MOO. What are a snowball's salient characteristics? What do you do with one? Well, you toss the snowball at someone else. But wait, before you do that, you first have to shape it, form it, pack the snow. Once you do toss it, do you still have it? No. So the program has to be able to distinguish between possession and nonpossession of the snowball. And maybe, if you hang onto it too long, it starts to melt. The exercise of thinking through what it takes to model a snowball in a believable fashion goes a long way toward capturing the appeal of what I mean by programming as world-making.

Grammarians may have noticed a familiar structure implicit in the snowball scenario I just sketched: We have a subject ("you," the maker of the snowball), an action (tossing it), and an object of the action (the snowball's target). From simple sentence structures made up of subjects, verbs, and objects we form stories. From stories, we make plots. Writers have long spoken of fiction as a kind of world-making; the novelist Ken Follett says on his Web site, "The basic challenge for the writer can be very simply explained—it is to create an imaginary world and then draw the reader into that imaginary world." The poet W.H. Auden famously referred to literature as "secondary worlds." Only now these worlds are being literalized—for example in Second Life, which boasts several million registered users, a functioning economy, and vast areas of virtual content. Second Life's residents are its true world-makers—everything one finds there, from vast virtual strip malls to neo-Victorian neighborhoods to fantastic and phantasmagoric costume balls, is user-created and user-contributed.

It used to be that we in English departments were fond of saying there was nothing outside of the text. Increasingly, though, texts take the form of worlds as much as words. Worlds are emerging as the consummate genre of the new century, whether it's the virtual worlds of Second Life or World of Warcraft or the more specialized venues seen in high-end simulation and visualization environments. Virtual worlds will be to the new century what cinema was to the last one and the novel to the century before that.

Importantly, "world" here means something very much like model, a selective and premeditated representation of reality, where some elements of the real are emphasized and exaggerated, others are distorted and caricatured, still others are absent altogether. Virtual worlds are interactive, manipulable, extensible; they are not necessarily games, though they may support and contain games alongside other systems. Virtual worlds are sites of exploration, simulation, play. We will want many virtual worlds, not few, because reality can be sliced and sampled in an infinite variety of ways.

All programming entails world-making, as the ritual act of writing and running Hello World reminds us. Virtual worlds simply lend literal and graphical form to this ideal. It's no accident that what was arguably the very first virtual world, Will Crowther's Colossal Cave Adventure, a text-only game programmed in 1975 depicting the user's exploration of a cave (it launched a whole genre of commercial successors), was embraced by programmers who saw unraveling the game's puzzles and tricky underground passages as a parable of their art.

Like computer programs in general, virtual worlds are empirical, but they are not objective. They embody their authors' biases, blind spots, ideologies, prejudices, and opinions. That is not a liability but their great asset, allowing them to serve as platforms for propagating some particular view of reality—what the game theorist and game designer Ian Bogost calls a "persuasive game," an example of which is his recent Fatworld, an unusual game about diet, exercise, and economics.

Playing Fatworld one quickly discovers that if you don't have enough virtual money, you must live in a virtual neighborhood whose amenities (or lack thereof) will have a tangible impact on your virtual waistline. Lacking a virtual car in this virtual world, you find that your dining options are limited mostly to fast food; you could walk to the virtual supermarket, but the nearest one is some ways away and it takes a lot of virtual time to get there. So you consume virtual junk food instead, which makes you virtually fat, which slows you down, which makes you even less likely to take a virtual walk to a neighborhood where you could find healthier virtual fare.

Is the real world really that simple? Of course not. Models draw their strength from selective representation. Ultimately what's at stake is not the kind of vocational computer literacy I was taught as an undergraduate, but what a growing number of practitioners in the digital humanities (and related disciplines, like digital art or game studies) have begun to call procedural rhetoric, or procedural literacy. In addition to Bogost, I'm thinking of recent work by Alex Galloway, Michael Mateas, Nick Montfort, Rita Raley, Stephen Ramsay, Noah Wardrip-Fruin, and Mackenzie Wark, all people who have at least one foot solidly planted in the humanities and who now write about these ideas at very high levels.

Procedural literacy, which starts with exercises like making a snowball, will be essential if humanities students are to understand virtual worlds as rhetorical and ideological spaces, just as film and the novel are likewise understood as forms of representation and rhetoric. One of the many legacies of the late Carnegie Mellon computer scientist Randy Pausch is a software learning environment called Alice, which teaches programming to nonspecialists precisely through the building of virtual worlds.

Computers should not be black boxes but understood as engines for creating powerful and persuasive models of the world around us.

COMPUTERS should not be black boxes but rather understood as engines for creating powerful and persuasive models of the world around us. The world around us (and inside us) is something we in the humanities have been interested in for a very long time. I believe that, increasingly, an appreciation of how complex ideas can be imagined and expressed as a set of formal procedures—rules, models, algorithms—in the virtual space of a computer will be an essential element of a humanities education. Our students will need to become more at ease reading (and writing) back and forth across the boundaries between natural and artificial languages. Such an education is essential if we are to cultivate critically informed citizens—not just because computers offer new worlds to explore, but because they offer endless vistas in which to see our own world reflected. ■

Matthew Kirschenbaum is an associate professor of English and associate director of the Maryland Institute for Technology in the Humanities at the University of Maryland at College Park.