



Like
Share
12k

Tweet
4,189

Email
/2538g1

225 (#disqus\_thread)

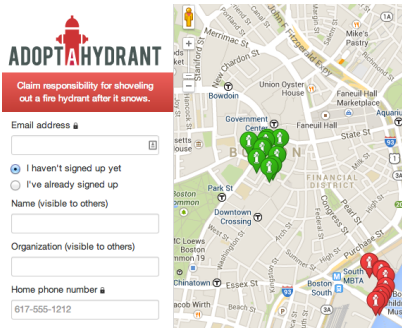

  
/PRINT
  
/2538g1

In the winter of 2011, a handful of software engineers landed in Boston just ahead of a crippling snowstorm. They were there as part of [Code for America](http://codeforamerica.org/) (<http://codeforamerica.org/>), a program that places idealistic young coders and designers in city halls across the country for a year. They'd planned to spend it building a new website for Boston's public schools, but within days of their arrival, the city all but shut down and the coders were stuck fielding calls in the city's snow emergency center.

In such snowstorms, firefighters can waste precious minutes finding and digging out hydrants. A city employee told the CFA team that the planning department had a list of street addresses for Boston's [13,000](http://www.cityofboston.gov/news/default.aspx?id=5444) hydrants. "We figured, 'Surely someone on the block with a shovel would volunteer if they knew where to look,'" says Erik Michaels-Ober, one of the CFA coders. So they got out their laptops.

Now, Boston has [adoptahydrant.org](http://adoptahydrant.org/) (<http://adoptahydrant.org/>), a simple website that lets residents "adopt" hydrants across the city. The site displays a map of little hydrant icons. Green ones have been claimed by someone willing to dig them out after a storm, red ones are still available—500 hydrants were adopted last winter.

Maybe that doesn't seem like a lot, but consider what the city pays to keep it running: \$9 a month in hosting costs. "I figured that even if it only led to a few fire hydrants being shoveled out, that could be the difference between life or death in a fire, so it was worth doing," Michaels-Ober says. And because the



Screenshot from Adopt-a-Hydrant Code for America

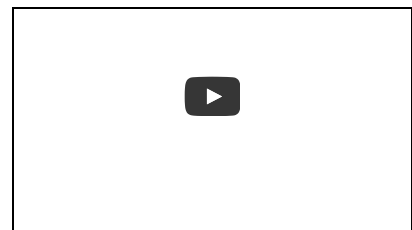
CFA team open-sourced the code, meaning they made it freely available for anyone to copy and modify, other cities can adapt it for practically pennies. It has been deployed in [Providence](http://adopt-a-hydrant-providence.herokuapp.com/) (<http://adopt-a-hydrant-providence.herokuapp.com/>), [Anchorage](http://http://ak-adopt-a-hydrant.herokuapp.com/) (<http://http://ak-adopt-a-hydrant.herokuapp.com/>), and [Chicago](http://www.adoptasidewalk.org/) (<http://www.adoptasidewalk.org/>). A Honolulu city employee heard about Adopt-a-Hydrant after cutbacks slashed his budget, and now Honolulu has [Adopt-a-Siren](http://http://sirens.honolulu.gov/) (<http://http://sirens.honolulu.gov/>), where volunteers can sign up to check for dead batteries in tsunami sirens across the city. In Oakland, it's [Adopt-a-Drain](http://adoptadrainoakland.com/) (<http://adoptadrainoakland.com/>).

Sounds great, right? These simple software solutions could save lives, and they were cheap and quick to build. Unfortunately, most cities will never get a CFA team, and most can't afford to keep a stable of sophisticated programmers in their employ, either. For that matter, neither can many software companies in Silicon Valley; the talent wars have gotten so bad that even brand-name tech firms have been forced to offer employees a bonus of upwards of \$10,000 if they help recruit an engineer.

In fact, even as the Department of Labor [predicts](http://www.bls.gov/emp/ep_table_102.htm) ([http://www.bls.gov/emp/ep\\_table\\_102.htm](http://www.bls.gov/emp/ep_table_102.htm)) the nation will add 1.2 million new computer-science-related jobs by 2022, we're graduating proportionately fewer ([http://nces.ed.gov/programs/digest/d13/tables/dt13\\_322.10.asp](http://nces.ed.gov/programs/digest/d13/tables/dt13_322.10.asp)) computer science majors than we did in the 1980s, and the number of students signing up for Advanced Placement computer science has [flatlined](http://media.collegeboard.com/digitalServices/pdf/research/2013/2013-Exam-Volume-Change.pdf) (<http://media.collegeboard.com/digitalServices/pdf/research/2013/2013-Exam-Volume-Change.pdf>).

There's a whole host of complicated reasons why, from boring curricula to a lack of qualified teachers to the fact that in most states computer science doesn't count toward graduation requirements. But should we worry? After all, anyone can learn to code after taking a few fun, interactive lessons at sites like [Codecademy](http://www.codecademy.com/) (<http://www.codecademy.com/>), as a flurry of articles in everything from *TechCrunch* to *Slate* have claimed. (Michael Bloomberg [pledged](https://twitter.com/MikeBloomberg/status/154999795159805952) (<https://twitter.com/MikeBloomberg/status/154999795159805952>) to enroll at Codecademy in 2012.) Twelve million people have [watched](https://code.org/news/12-million-youtube-views) (<https://code.org/news/12-million-youtube-views>) a video from Code.org in which celebrities like NBA All-Star Chris Bosh and will.i.am pledged to spend an hour learning code, a notion endorsed by President Obama, who [urged](http://www.whitehouse.gov/blog/2013/12/09/don-t-just-play-your-phone-program-it) (<http://www.whitehouse.gov/blog/2013/12/09/don-t-just-play-your-phone-program-it>) the nation: "Don't just play on your phone—program it."

So you might be forgiven for thinking that learning code is a short, breezy ride



---

Code.org

to a lush startup job with a foosball table and free kombucha, especially given all the hype about billion-dollar companies launched by self-taught wunderkinds (with nary a mention of the private tutors and coding camps that helped some of them get there). The truth is, code—if what we're talking about is the chops you'd need to qualify for a programmer job—is hard, and lots of people would find those jobs tedious and boring.

But let's back up a step: What if learning to code weren't actually the most important thing? It turns out that rather than increasing the number of kids who can crank out thousands of lines of JavaScript, we first need to boost the number who understand what code can do. As the cities that have hosted Code for America teams will tell you, the greatest contribution the young programmers bring isn't the software they write. It's the way they *think*. It's a principle called "computational thinking," and knowing all of the Java syntax in the world won't help if you can't think of good ways to apply it.

Unfortunately, the way computer science is currently taught in high school tends to throw students into the programming deep end, reinforcing the notion that code is just for coders, not artists or doctors or librarians. But there is good news: Researchers have been experimenting with new ways of teaching computer science, with intriguing results. For one thing, they've seen that leading with computational thinking instead of code itself, and helping students imagine how being computer savvy could help them in any career, boosts the number of girls and kids of color taking—and sticking with—computer science. Upending our notions of what it means to interface with computers could help democratize the biggest engine of wealth since the Industrial Revolution.

**SO WHAT IS COMPUTATIONAL THINKING?** If you've ever improvised dinner, pat yourself on the back: You've engaged in some light CT.

There are those who open the pantry to find a dusty bag of legumes and some sad-looking onions and think, "Lentil soup!" and those who think, "Chinese takeout." A practiced home cook can mentally sketch the path from raw ingredients to a hot meal, imagining how to substitute, divide, merge, apply external processes (heat, stirring), and so on until she achieves her end. Where the rest of us see a dead end, she sees the potential for something new.

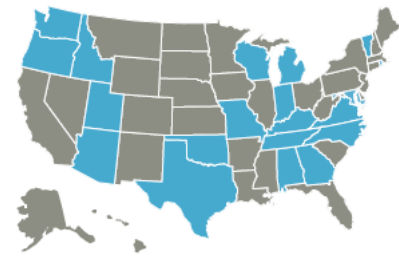
If seeing the culinary potential in raw ingredients is like computational thinking, you might think of a software algorithm as a kind of recipe: a step-by-step guide on how to take a bunch of random ingredients and start layering them together in certain quantities, for certain amounts of time, until they

## THE PIPELINE PROBLEM

Among AP courses taken last year, computer science is near the bottom.

English	862,000
Calculus	387,000
Spanish	154,000
Chemistry	140,000
European history	110,000
Computer science	31,000

**Only 20 states** count computer science toward graduation requirements in math or science.



Sources: College Board, Computer Science Teachers Association

produce the outcome you had in mind.

Like a good algorithm, a good recipe follows some basic principles.

Ingredients are listed first, so you can collect them before you start, and there's some logic in the way they are listed: olive oil before cumin because it goes in the pan first. Steps are presented in order, not a random jumble, with staggered tasks so that you're chopping veggies while waiting for water to boil. A good recipe spells out precisely what size of dice or temperature you're aiming for. It tells you to look for signs that things are working correctly at each stage—the custard should coat the back of a spoon. Opportunities for customization are marked—use twice the milk for a creamier texture—but if any ingredients are absolutely crucial, the recipe makes sure you know it. If you need to do something over and over—add four eggs, one at a time, beating after each—those tasks are boiled down to one simple instruction.

Much like cooking, computational thinking begins with a feat of imagination, the ability to envision how digitized information—ticket sales, customer addresses, the temperature in your fridge, the sequence of events to start a car engine, anything that can be sorted, counted, or tracked—could be combined and changed into something new by applying various computational techniques. From there, it's all about "decomposing" big tasks into a logical series of smaller steps, just like a recipe.

Those techniques include a lot of testing along the way to make sure things are working. The culinary principle of *[mise en place](http://www.reluctantgourmet.com/mise-en-place/)*

(<http://www.reluctantgourmet.com/mise-en-place/>) is akin to the computational

principle of sorting: organize your data first, and you'll cut down on search

time later. Abstraction ([http://www.google.com/edu/computational-thinking/what-is-](http://www.google.com/edu/computational-thinking/what-is-ct.html#decomposition)

[ct.html#decomposition](http://www.google.com/edu/computational-thinking/what-is-ct.html#decomposition)) is like the concept of "mother sauces" in French cooking

(béchamel, tomato, hollandaise), building blocks to develop and reuse in

hundreds of dishes. There's iteration: running a process over and over until

you get a desired result. The principle of parallel processing

([http://gribblelab.org/CBootcamp/A2\\_Parallel\\_Programming\\_in\\_C.html](http://gribblelab.org/CBootcamp/A2_Parallel_Programming_in_C.html)) makes use of all

available downtime (think: making the salad while the roast is cooking). Like a

good recipe, good software is really clear about what you can tweak and what

you can't. It's explicit. Computers don't get nuance; they need everything

spelled out for them.

Put another way: Not every cook is a David Chang (<http://momofuku.com/team/>),

not every writer is a Jane Austen, and not every computational thinker is a

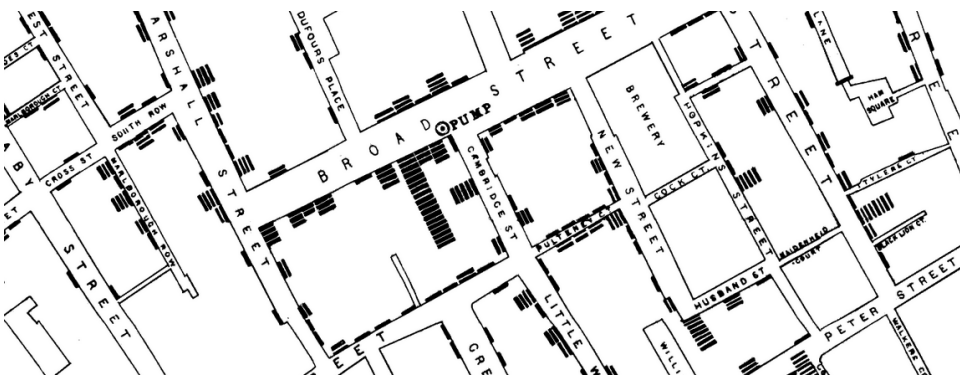
Guido van Rossum, the inventor (<https://www.python.org/~guido/>) of the influential

Python programming language. But just as knowing how to scramble an egg or

write an email makes life easier, so too will a grasp of computational thinking. Yet the "learn to code!" camp may have set people on the uphill path of mastering C++ (<http://www.cplusplus.com/>) syntax instead of encouraging all of us to think a little more computationally.

The happy truth is, if you get the fundamentals about how computers think, and how humans can talk to them in a language the machines understand, you can imagine a project that a computer could do, and discuss it in a way that will make sense to an actual programmer. Because as programmers will tell you, the building part is often not the hardest part: It's figuring out what to build. "Unless you can think about the ways computers can solve problems, you can't even know how to ask the questions that need to be answered," says [Annette Vee](http://www.annettevee.com/) (<http://www.annettevee.com/>), a University of Pittsburgh professor who studies the spread of computer science literacy.

Indeed, some powerful computational solutions take just a few lines of code—or no code at all. Consider this lo-fi example: In 1854 (<http://www.jsi.com/JSIInternet/About/snow.cfm>), a London physician named John Snow helped (<http://www.albany.edu/faculty/fboscoe/papers/koch2009.pdf>) squelch a cholera outbreak that had killed 616 residents. Brushing aside the prevailing theory of the disease—deadly miasma—he surveyed relatives of the dead about their daily routines. A map ([http://www.ph.ucla.edu/epi/snow/snowmap1\\_1854.html](http://www.ph.ucla.edu/epi/snow/snowmap1_1854.html)) he made connected the disease to drinking habits: tall stacks of black lines, each representing a death, grew around a water pump on Broad Street in Soho that happened to be near a leaking cesspool. His theory: The disease was in the water. Classic principles of computational thinking came into play here, including merging two datasets to reveal something new (locations of deaths plus locations of water pumps), running the same process over and over and testing the results, and pattern recognition. The pump was closed, and the outbreak subsided.



Detail from "On the Mode of Communication of Cholera," 1854

Or take Adopt-a-Hydrant. Under the hood, it isn't a terribly sophisticated piece of software. What's ingenious is simply that someone knew enough to

say: Here's a database of hydrant locations, here is a universe of people willing to help, let's match them up. The computational approach is rooted in seeing the world as a series of puzzles, ones you can break down into smaller chunks and solve bit by bit through logic and deductive reasoning. That's why Jeannette Wing (<http://www.cs.cmu.edu/~fox/foxnet/people/wing/>), a VP of research at Microsoft who popularized (<http://www.cs.cmu.edu/afs/cs/usr/wing/www/publications/Wing06.pdf>) the term "computational thinking," says it's a shame to think CT is just for programmers. "Computational thinking involves solving problems, designing systems, and understanding human behavior," she writes in a publication of the Association for Computing Machinery. Those are handy skills for everybody, not just computer scientists.

In other words, computational thinking opens doors. For while it may seem premature to claim that today every kid needs to code, it's clear that they're increasingly surrounded by opportunities to code—opportunities that the children of the privileged are already seizing. The parents of Facebook founder Mark Zuckerberg got (<http://news.blogs.cnn.com/2012/05/17/timeline-mark-zuckerbergs-rise-from-child-prodigy-to-facebook-billionaire/>) him ([http://books.google.com/books?id=YrZ1FLpHWHIC&pg=PA2005&lpg=PA2005&dq=mark+zuckerberg+tutor+dauid+newman&source=bl&ots=hunBoGbWDj&sig=wdFVm5spzYRN2vs\\_iw5c-L7crCM&hl=en&sa=X&ei=ljtpU4TLM43YoASmYLwCg&ved=0CEcQ6AEwBzgK#v=onepage&q=mark%20zuckerberg%20tutor%20dauid%20newman&f=false](http://books.google.com/books?id=YrZ1FLpHWHIC&pg=PA2005&lpg=PA2005&dq=mark+zuckerberg+tutor+dauid+newman&source=bl&ots=hunBoGbWDj&sig=wdFVm5spzYRN2vs_iw5c-L7crCM&hl=en&sa=X&ei=ljtpU4TLM43YoASmYLwCg&ved=0CEcQ6AEwBzgK#v=onepage&q=mark%20zuckerberg%20tutor%20dauid%20newman&f=false)) a private computer tutor when he was in middle school. Last year, 13,000 people chipped in more than (<https://www.kickstarter.com/projects/danshapiro/robot-turtles-the-board-game-for-little-programmer>) \$600,000 via Kickstarter for their own limited-edition copy of Robot Turtles, a board game that teaches programming basics to kids as young as three. There are plenty of free, kid-oriented code-learning sites—like Scratch (<http://scratch.mit.edu/>), a programming language for children developed at MIT—but parents and kids in places like San Francisco or Austin are more likely to know they exist.

Computer scientists have been warning for decades that understanding code will one day be as essential as reading and writing. If they're right, understanding the importance of computational thinking can't be limited to the elite, not if we want some semblance of a democratic society. Self-taught auteurs will always be part of the equation, but to produce tech-savvy citizens "at scale," to borrow an industry term, the heavy lifting will happen in public school classrooms. Increasingly, to have a good shot at a good job, you'll need to be code literate.

---

# Upending our notions of what it means to interface with computers could help democratize the biggest engine of wealth since the Industrial Revolution.

---

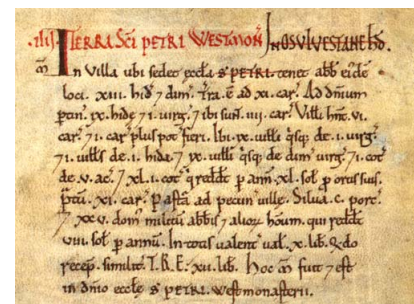
"**CODE LITERATE.**" Sounds nice, but what does it mean? And where does literacy end and fluency begin? The best way to think about that is to look to the history of literacy itself.

Reading and writing have become what researchers have called "interiorized" or "infrastructural," a technology baked so deeply into everyday human life that we're never surprised to encounter it. It's the main medium through which we connect, via not only books and papers, but text messages and the voting booth, medical forms and shopping sites. If a child makes it to adulthood without being able to read or write, we call that a societal failure.

Yet for thousands of years writing was the preserve of the professional scribes employed by the elite. So what moved it to the masses? In Europe at least, [writes](http://licsjournal.org/OJS/index.php/LiCS/article/view/24) (<http://licsjournal.org/OJS/index.php/LiCS/article/view/24>) literacy researcher Vee, the tipping point was the Domesday Book (<http://www.nationalarchives.gov.uk/domesday/>), an 11th-century survey of landowners that's been called the oldest public record in England.

Commissioned by William the Conqueror to take stock of what his new subjects held in terms of acreage, tenants, and livestock so as to better tax them, royal scribes fanned across the countryside taking detailed notes during in-person interviews. It was like a hands-on demo on the efficiencies of writing, and it proved contagious. Despite skepticism—writing was hard, and maybe involved black magic—other institutions started putting it to use. Landowners and vendors required patrons and clients to sign deeds and receipts, with an "X" if nothing else. Written records became admissible in court. Especially once Johannes Gutenberg (<http://web.mit.edu/invent/iow/gutenberg.html>) invented the printing press, writing seeped into more and more aspects of life, no longer a rarefied skill restricted to a cloistered class of aloof scribes but a function of everyday society.

Fast forward to 19th-century America, and it'd be impossible to walk down a street without being bombarded with written information, from newspapers to street signs to store displays; in the homes of everyday people, personal letters and account ledgers could be found. "The technology of writing became infrastructural," Vee writes in her paper "Understanding Computer



A page from the  
Domesday Book National  
Archives, UK

Programming As a Literacy." "Those who could not read text began to be recast as 'illiterate' and power began to shift towards those who could."

Teaching children how to read became a civic and moral imperative. Literacy rates soared over the next century, fostered through religious campaigns, the nascent public school system, and the at-home labor of many mothers.

Of course, not everyone was invited in immediately: Illiteracy among women, slaves, and people of color was often outright encouraged, sometimes even legally mandated. But today, while only some consider themselves to be "writers," practically everybody reads and writes every day. It's hard to imagine there was ever widespread resistance to universal literacy.

So how does the history of computing compare? Once again, says Vee, it starts with a census. In 1880, a Census Bureau statistician, Herman Hollerith

([http://www.census.gov/history/www/census\\_then\\_now/notable\\_alumni/herman\\_hollerith.html](http://www.census.gov/history/www/census_then_now/notable_alumni/herman_hollerith.html))

, saw that the system of collecting and sorting surveys by hand was buckling under the weight of a growing population. He devised an electric tabulating machine, and generations of these "Hollerith machines

([https://www.census.gov/history/www/innovations/technology/the\\_hollerith\\_tabulator.html](https://www.census.gov/history/www/innovations/technology/the_hollerith_tabulator.html)) "

were used by the bureau until the 1950s, when the first commercial mainframe, the UNIVAC

([https://www.census.gov/history/www/innovations/technology/univac\\_i.html](https://www.census.gov/history/www/innovations/technology/univac_i.html)), was developed

with a government research grant. "The first successful civilian computer," it was a revolution in computing technology: Unlike the "dumb" Hollerith machine and its cousins, which ran on punch cards, vacuum tubes, and other mechanical inputs that had to be manually entered over and over again, the UNIVAC had memory. It could store instructions, in the form of programs, and remember earlier calculations for later use.<sup>1</sup>

Once the UNIVAC was unveiled, research institutions and the private sector began clamoring (<http://pabook.libraries.psu.edu/palitmap/UNIVAC.html>) for mainframes of their own. The scribes of the computer age, the early programmers who had worked on the first large-scale computing projects for the government during the war, took jobs at places like Bell Labs, the airline industry, banks, and research universities. "The spread of the texts from the central government to the provinces is echoed in the way that the programmers who cut their teeth on major government-funded software projects then circulated out into smaller industries, disseminating their knowledge of code writing further," Vee writes. Just as England had gone from oral tradition to written record after the Domesday Book, the United States in the 1960s and '70s shifted from written to computational record.

<sup>1</sup> The evolution of communication technologies has always been an issue of memory. For thousands of years, the oral tradition had enough storage space to house the expanse of human records and information. As communities got bigger, oral tradition started maxing out. So a new technology sprang up, one that could distill thought into a series of symbolic scratches that could be packaged up, transported, and recompiled by the user into language and thought. But while books have immensely greater RAM than a song poem, a computer offers exponentially more capacity



The 1980s made computers personal, and today it's impossible not to engage in conversations powered by code, albeit code that's hidden beneath the interfaces of our devices. But therein lies a new problem: The easy interface creates confusion around what it means to be "computer literate." Interacting with an app is very different from making or tweaking or understanding one, and opportunities to do the latter remain the province of a specialized elite. In many ways, we're still in the "scribal stage" of the computer age.

But the tricky thing about literacy, Vee says, is that it begets more literacy. It happened with writing: At first, laypeople could get by signing their names with an "X." But the more people used reading and writing, the more was required of them.

We can already see code leaking into seemingly far-removed fields. Hospital specialists (<http://www.nsf.gov/cise/csbytes/newsletter/vol2/pdf/vol2i2.pdf>) collect data from the heartbeat monitors of day-old infants, and run algorithms to spot babies likely to have respiratory failure. Netflix is a gigantic experiment in statistical machine learning. Legislators are being challenged to understand encryption ([http://www.washingtonpost.com/world/national-security/at-senate-hearing-nsa-director-defends-spying-program/2013/12/11/a21ae186-62b1-11e3-aa81-e1dab1360323\\_story.html](http://www.washingtonpost.com/world/national-security/at-senate-hearing-nsa-director-defends-spying-program/2013/12/11/a21ae186-62b1-11e3-aa81-e1dab1360323_story.html)) and relational databases during hearings (<http://thehill.com/policy/technology/196711-house-panel-to-hold-very-comprehensive-nsa-hearing>) on the NSA.

The most exciting advances in most scientific and technical fields already involve big datasets, powerful algorithms, and people who know how to work with both. But that's increasingly true in almost any profession. English literature and computer science researchers fed Agatha Christie's oeuvre into a computer, ran a textual-analysis program, and discovered (<http://ftp.cs.toronto.edu/pub/gh/Lancashire+Hirst-extabs-2009.pdf>) that her vocabulary shrank significantly in her final books. They drew from the work of brain researchers and put forth a new hypothesis: Christie suffered from Alzheimer's. "More and more, no matter what you're interested in, being computationally savvy will allow you to do a better job," says (<http://www.ncwit.org/profile/jan-cuny>) Jan Cuny, a leading CS researcher at the National Science Foundation (NSF).



**Grace Hopper led the team that developed the UNIVAC, the first commercial computer.** Smithsonian Institution

It may be hard to swallow the idea that coding could ever be an everyday activity on par with reading and writing in part because it looks so foreign (what's with all the semicolons and carets)? But remember that it took hundreds of years to settle on the writing conventions we take for granted today: Early spellings of words—*Whan that Aprille with his shoures soote* (<http://www.bartleby.com/40/0101.html>)—can seem as foreign to modern readers as today's code snippets do to nonprogrammers. Compared to the thousands of years writing has had to go from notched sticks to glossy magazines, digital technology has, in 60 years, evolved exponentially faster.

Our elementary-school language arts teachers didn't drill the alphabet into our brains anticipating Facebook or WhatsApp or any of the new ways we now interact with written material. Similarly, exposing today's third-graders to a dose of code may mean that at 30 they retain enough to ask the right questions of a programmer, working in a language they've never seen on a project they could never have imagined.

---

## To produce tech-savvy citizens "at scale," to borrow an industry term, the heavy lifting will happen in public school classrooms

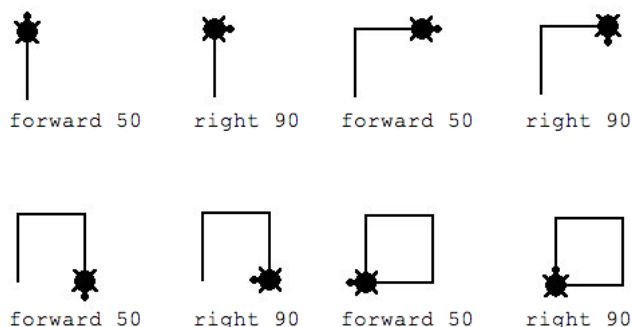
---

**ONE DAY LAST YEAR**, Neil Fraser, a young software engineer at Google, showed up (<https://neil.fraser.name/news/2013/03/16/>) unannounced at a primary school in the coastal Vietnamese city of Da Nang. Did the school have

computer classes, he wanted to know, and could he sit in? A school official glanced at Fraser's Google business card and led him into a classroom of fifth-graders paired up at PCs while a teacher looked on. What Fraser saw on their screens came as a bit of a shock.

Fraser, who was in Da Nang visiting his girlfriend's family, works in Google's education department in Mountain View, teaching JavaScript to new recruits. His interest in computer science education often takes him to high schools around the Bay Area, where he tells students that code is fun and interesting, and learning it can open doors after graduation.

The fifth-graders in Da Nang were doing exercises in Logo, a simple program developed at MIT in the 1970s to introduce children to programming. A turtle-shaped avatar blinked on their screens and the kids fed it simple commands in Logo (<http://el.media.mit.edu/logo-foundation/logo/index.html>)'s language, making it move around, leaving a colored trail behind. Stars, hexagons, and ovals bloomed on the monitors.



---

**Simple commands in Logo.** MIT Media Lab

Fraser, who learned Logo when the program was briefly popular in American elementary schools, recognized the exercise. It was a lesson in loops, a bedrock programming concept in which you tell the machine to do the same thing over and over again, until you get a desired result. "A quick comparison with the United States is in order," Fraser wrote later in a blog post. At Galileo Academy, San Francisco's magnet school for science and technology, he'd found juniors in a computer science class struggling with the concept of loops. The fifth-graders in Da Nang had outpaced upperclassmen at one of the Bay Area's most tech-savvy high schools.

Another visit to an 11th-grade classroom in Ho Chi Minh City revealed students coding their way through a logic puzzle embedded in a digital maze. "After returning to the US, I asked a senior engineer how he'd rank this question on a Google interview," Fraser wrote. "Without knowing the source of the question, he judged that this would be in the top third."

Early code education isn't just happening in Vietnamese schools. Estonia, the [birthplace](http://www.microsoft.com/en-us/news/stories/skype/skype-chapter-1-skype-at-10.aspx) of Skype, rolled out [a countrywide](http://www.wired.com/2012/09/estonia-reprograms-first-graders-as-web-coders/) programming-centric curriculum for students as young as six in 2012. In September, the United Kingdom will [launch](https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/210969/NC_framework_document_-_FINAL.pdf) a mandatory computing syllabus for all students ages 5 to 16.

Meanwhile, even as US enrollment in almost all other STEM (science, technology, engineering, and math) fields has [grown](http://nces.ed.gov/nationsreportcard/pdf/studies/2011462.pdf) over the last 20 years, computer science has actually *lost* students, dropping from 25 percent of high school students earning credits in computer science to only 19 percent by 2009, [according to](http://nces.ed.gov/nationsreportcard/pdf/studies/2011462.pdf) the National Center for Education Statistics.

"Our kids are competing with kids from countries that have made computer science education a No. 1 priority," says Chris Stephenson, the former head of the [Computer Science Teachers Association](http://csta.acm.org/) (CSTA). Unlike countries with federally mandated curricula, in the United States computer lesson plans can vary widely between states and even between schools in the same district. "It's almost like you have to go one school at a time," Stephenson says. In fact, currently only 20 states and Washington, DC, allow computer science to count toward core graduation requirements in math or science, and not one requires students to take a computer science course to graduate. Nor do the new [Common Core](http://www.corestandards.org/about-the-standards/) standards, a push to make K-12 curricula more uniform across states, include computer science requirements.

It's no surprise, then, that the AP computer science course is among the College Board's least popular offerings; [last year](http://media.collegeboard.com/digitalServices/pdf/research/2013/2013-Exam-Volume-Change.pdf) almost four times more students tested in geography ([114,000](http://media.collegeboard.com/digitalServices/pdf/research/2013/2013-Exam-Volume-Change.pdf)) than computer science ([31,000](http://media.collegeboard.com/digitalServices/pdf/research/2013/2013-Exam-Volume-Change.pdf)). And most kids don't even get to make that choice; [only 17](http://media.collegeboard.com/digitalServices/pdf/research/2013/Program-Summary-Report-)

2013.pdf) percent of US high schools that have advanced placement courses

(<http://media.collegeboard.com/digitalServices/pdf/research/2013/2013-Annual-Participation.pdf>)

do so in CS. It was 20 percent in 2005.

For those who do take an AP computer science class—a yearlong course in

Java (<http://www.universityhighschool.org/academics/apcourses/>), which is sort of like

teaching cooking by showing how to assemble a KitchenAid—it won't count

toward core graduation requirements in most states. What's more, many

counselors see AP CS as a potential GPA ding, and urge students to load up on

known quantities like AP English or US history. "High school kids are

overloaded already," says Joanna Goode (<https://education.uoregon.edu/users/goode>),

a leading researcher at the University of Oregon's education department, and

making time for courses that don't count toward anything is a hard sell.

In any case, it's hard to find anyone to teach these classes

([http://csta.acm.org/ComputerScienceTeacherCertification/sub/CSTA\\_BugsInTheSystem.pdf](http://csta.acm.org/ComputerScienceTeacherCertification/sub/CSTA_BugsInTheSystem.pdf)).

Unlike fields such as English and chemistry, there isn't a standard path

([http://csta.acm.org/ComputerScienceTeacherCertification/sub/CSTA\\_BugsInTheSystem.pdf](http://csta.acm.org/ComputerScienceTeacherCertification/sub/CSTA_BugsInTheSystem.pdf)) for

aspiring CS teachers in grad school or continuing education programs. And

thanks to wildly inconsistent certification rules between states, certified CS

teachers can get stuck teaching math or library sciences if they move.

Meanwhile, software whizzes often find the lure of the startup salary much

stronger than the call of the classroom, and anyone who tires of Silicon Valley

might find that its "move fast and break things" mantra doesn't transfer neatly

to pedagogy.

And while many kids have mad skills in movie editing or Photoshopping, such

talents can lull parents into thinking they're learning real computing. "We

teach our kids how to be consumers of technology, not creators of technology,"

notes the NSF's Cuny.

Or, as Cory Doctorow, an editor of the technology-focused blog *Boing Boing*,

put it in a manifesto ([http://boingboing.net/2010/04/02/why-i-wont-buy-an-ipad-and-think-](http://boingboing.net/2010/04/02/why-i-wont-buy-an-ipad-and-think-vo.html)

[vo.html](http://boingboing.net/2010/04/02/why-i-wont-buy-an-ipad-and-think-vo.html)) titled "Why I Won't Buy an iPad": "Buying an iPad for your kids isn't a

means of jump-starting the realization that the world is yours to take apart

and reassemble; it's a way of telling your offspring that even changing the

batteries is something you have to leave to the professionals."

But school administrators know that gleaming banks of shiny new machines

go a long way in impressing parents and school boards. Last summer

(<http://www.latimes.com/local/la-me-lausd-ipads-20130828.0.906926.story#axzz2zdleRjUx>),

the Los Angeles Unified School District set aside

(<http://articles.latimes.com/2013/jun/18/local/la-me-ln-lausd-chooses-ipads-for-pilot-20130618>) a

billion dollars ([http://www.latimes.com/local/la-me-ipad-probe-](http://www.latimes.com/local/la-me-ipad-probe-20140422.0.6764867.story#axzz2zdM96X3G)

20140422.0.6764867.story#axzz2zdM96X3G) to buy an iPad for all 640,000

([http://home.lausd.net/apps/pages/index.jsp?uREC\\_ID=178745&type=d&pREC\\_ID=371201](http://home.lausd.net/apps/pages/index.jsp?uREC_ID=178745&type=d&pREC_ID=371201))

children in the district. To pay for the program, the district dipped into school construction bonds. Still, some parents and principals told

(<http://www.latimes.com/local/la-me-lausd-ipads-20130828.0.906926.story#axzz2zdleRjUx>) the

*Los Angeles Times* they were thrilled about it. "It gives us the sense of hope that these kids are being looked after," said one parent.<sup>2</sup>

Sure, some schools are woefully behind on the hardware equation, but

according to (<http://nces.ed.gov/fastfacts/display.asp?id=46>) a 2010 federal study, only

3 percent of teachers nationwide lacked daily access to a computer in their

classroom, and the nationwide ratio of students to school computers was a

little more than 5-to-1. As to whether kids have computers at home—that

doesn't seem to make much difference in overall performance, either. A study

(<http://www.nber.org/papers/w19060.pdf>) from the National Bureau of Economic

Research reviewed the grades, test scores, homework, and attendance of

California 6th- to 10th-graders who were randomly given computers to use at

home for the first time. A year later, the study found, nothing happened. Test

scores, grades, disciplinary actions, time spent on homework: None of it went

up or down—except the kids did log a lot more time playing games.

<sup>2</sup> *The kids did quickly learn to hack their iPads, so there's some hope for actual inventiveness.*

---

## We're still in the "scribal stage" of the computer age, where skills are in the hands of an elite.

---

**ONE SUNNY MORNING** last summer, 40 Los Angeles teachers sat in a warm classroom at UCLA playing with crayons, flash cards, and Legos. They were students again for a week, at a workshop on how to teach computer science. Which meant that first they had to learn computer science.

The lesson was in binary numbers, or how to write any number using just two digits. "Computers can only talk in ones and zeros," explained the instructor, a fellow teacher who'd taken the same course. The course is funded by the National Science Foundation, and so is the experimental new blueprint it trains teachers to use, called Exploring Computer Science

(<http://www.exploringcs.org/>) (ECS). "You gotta talk to them in their language."

Made sense at first, but when it came to turning the number 1,250 into binary,

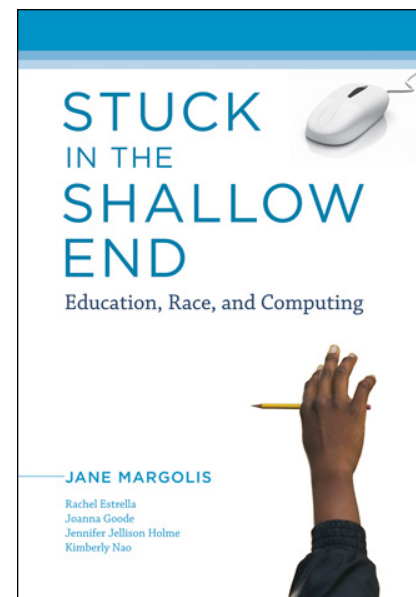
the class started falling apart. At one table, two female teachers politely endured a long, wrong explanation from an older male colleague. A teacher behind them mumbled, "I don't get it," pushed his flash cards away, and counted the minutes to lunchtime. A table of guys in their 30s was loudly sprinting toward an answer, and a minute later the bearded white guy at the head of their table, i.e., the one most resembling a classic programmer, shot his hand up with the answer and an explanation of how he got there:

"Basically what you do is, you just turn it into an algorithm." Blank stares suggested few colleagues knew what an algorithm was—in this case a simple, step-by-step process for turning a number into binary. (The answer, if you're curious, is 010011100010 (<http://www.binary-code.org/binary/12bit/010011100010/>).)

This lesson—which by the end of the day clicked for most in the class—might seem like most people's image of CS, but the course these teachers are learning to teach couldn't look more different from classic AP computer science. Much of what's taught in ECS is about the why of computer science, not just the how. There are discussions and writing assignments on everything from personal privacy in the age of Big Data to the ethics of robot labor to how data analysis could help curb problems like school bullying. Instead of rote Java learning, it offers lots of logic games and puzzles that put the focus on computing, not computers. In fact, students hardly touch a computer for the first 12 weeks.

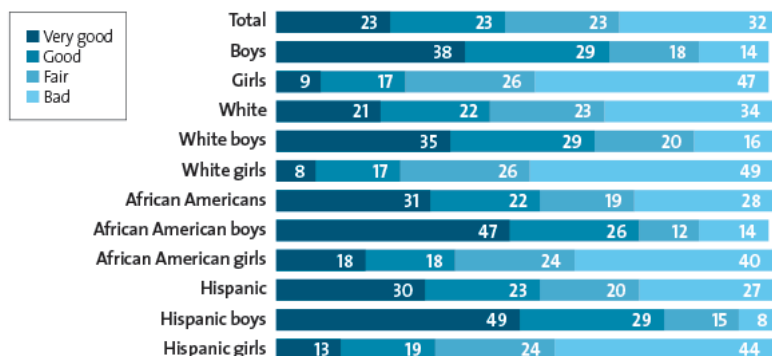
"Our curriculum doesn't lead with programming or code," says Jane Margolis (<http://gseis.ucla.edu/directory/jane-margolis/>), a senior researcher at UCLA who helped design the ECS curriculum and whose book *Stuck in the Shallow End: Education, Race, and Computing* (<http://www.amazon.com/Stuck-Shallow-End-Education-Computing/dp/0262514044>) provides much of the theory behind the lesson plans. "There are so many stereotypes associated with coding, and often it doesn't give the broader picture of what the field is about. The research shows you want to contextualize, show how computer science is relevant to their lives." ECS lessons ask students to imagine how they'd make use of various algorithms as a chef, or a carpenter, or a teacher, how they could analyze their own snack habits to eat better, and how their city council could use data to create cleaner, safer streets.

The ECS curriculum (<http://www.exploringcs.org/curriculum>) is now offered to 2,400 students (<http://www.exploringcs.org/about/results>) at 31 Los Angeles public high schools and a smattering of schools in other cities, notably Chicago and Washington, DC. Before writing it, Margolis and fellow researchers spent (<http://www.exploringcs.org/about/the-research-behind-ecs>) three years visiting schools across the Los Angeles area—overcrowded urban ones and plush suburban



ones—to understand why few girls and students of color were taking computer science. At a tony school in West LA that the researchers dubbed "Canyon Charter High," they noticed students of color traveling long distances to get to school, meaning they couldn't stick around for techie extracurriculars or to simply hang out with like-minded students.

Percentage of students who consider a career in computers a "good choice" for someone like them:



Association for Computing Machinery, WGBH Educational Foundation

Mother Jones

Equally daunting were the stereotypes. Take Janet, the sole black girl in Canyon's AP computer science class, who told the researchers she signed up for the course in part "because we [African American females] were so limited in the world, you know, and just being able to be in a class where I can represent who I am and my culture was really important to me." When she had a hard time keeping up—like most kids in the class—the teacher, a former software developer who, researchers noted, tended to let a few white boys monopolize her attention, pulled Janet aside and suggested she drop the class, explaining that when it comes to computational skills, you either "have it or don't have it."

Research shows that girls tend to pull away from STEM subjects—including computer science—around middle school, while rates of boys in these classes stay steady. Fortunately, says Margolis, there's evidence that tweaking the way computer science is introduced can make a difference. A [2009 study](http://www.acm.org/membership/NIC.pdf) (<http://www.acm.org/membership/NIC.pdf>) tested various messages about computer science with college-bound teens. It found that explaining how programming skills can be used to "do good"—connect with one's community, make a difference on big social problems like pollution and health care—reverberated strongly with girls. Far less successful were messages about getting a good job or being "in the driver's seat" of technological innovation—i.e., the dominant cultural narratives about why anyone would learn to code.

"For me, computer science can be used to implement social change," says [Kim Merino](http://ampersand.gseis.ucla.edu/tag/kim-merino/) (<http://ampersand.gseis.ucla.edu/tag/kim-merino/>), a self-described "social-

What word comes to mind when you see or hear the word "computing"?

Girls:

"typing"

"math"

"boredom"

"nerd"

Boys:

"video games"

"design"

"electronics"

"solving problems"

"interesting"

Association for Computing Machinery, WGBH Educational Foundation

Mother Jones



justice-obsessed queer Latina nerd history teacher" who decided to take the ECS training a couple of years ago. Now, she teaches the class to middle and high schoolers at the UCLA Community School, an experimental new public K-12 school. "I saw this as a new frontier in the social-justice fight," she says. "I tell my students, 'I don't necessarily want to teach you how to get rich. I want to teach you to be a good citizen.'"

Merino's father was an aerospace engineer for Lockheed Martin. So you might think adapting to CS would be easy for her. Not quite. Most of the teachers she trained with were men. "Out of seven women, there were two of color.

Honestly, I was so scared. But now, I take that to my classroom. At this point my class is half girls, mostly Latina and Korean, and they still come into my class all nervous and intimidated. My job is to get them past all of that, get them excited about all the things they could do in their lives with programming."

Merino has spent the last four years teaching kids of color growing up in inner cities to imagine what they could do with programming—not as a replacement for, but as part of their dreams of growing up to be doctors or painters or social workers. But Merino's partner's gentle ribbings about how they'd ever start a family on a teacher's salary eventually became less gentle. She just took a job as director of professional development at [CodeHS \(http://codehs.com/\)](http://codehs.com/), an educational startup in San Francisco.

---

---

## "We teach our kids how to be consumers of technology, not creators of technology."

---

---

**IT WAS A LITTLE MORE THAN** a century ago that literacy became universal in Western Europe and the United States. If computational skills are on the same trajectory, how much are we hurting our economy—and our democracy—by not moving faster to make them universal?

There's the talent squeeze, for one thing. Going by the number of computer science majors graduating each year, we're producing less than half of the talent needed to fill the Labor Department's job projections. [Women \(http://www.bls.gov/cps/cpsaat11.pdf\)](http://www.bls.gov/cps/cpsaat11.pdf) currently make up [20 percent of the software workforce, blacks and Latinos around 5 percent each. Getting more of them in the computing pipeline is simply good business sense.](http://www.census.gov/prod/2013pubs/acs-24.pdf)

It would also create a future for computing that more accurately reflects its past. A female mathematician named [Ada Lovelace](http://www.computerhistory.org/babbage/adalovelace/) (<http://www.computerhistory.org/babbage/adalovelace/>) wrote the first algorithm ever intended to be executed on a machine in 1843. The term "programmer" was used during World War II to [describe the women](http://www.eg.bucknell.edu/~csci203/2014-spring/hw/hwo6/assets/womenOfENIAC.pdf) (<http://www.eg.bucknell.edu/~csci203/2014-spring/hw/hwo6/assets/womenOfENIAC.pdf>) who worked on the world's first large-scale electronic computer, the [ENIAC](http://www.seas.upenn.edu/about-seas/eniac/) (<http://www.seas.upenn.edu/about-seas/eniac/>) machine, which used calculus to come up with tables to improve artillery accuracy<sup>3</sup>. In 1949, [Rear Adm. Grace Hopper](http://www.history.navy.mil/bios/hopper_grace.htm) ([http://www.history.navy.mil/bios/hopper\\_grace.htm](http://www.history.navy.mil/bios/hopper_grace.htm)) helped develop the UNIVAC, the first general-purpose computer, a.k.a. a mainframe, and in 1959 her work led to the development of COBOL, the first programming language written for commercial use.

Excluding huge swaths of the population also means prematurely killing off untold ideas and innovations that could make everyone's lives better. Because while the rash of meal delivery and dating apps designed by today's mostly young, male, urban programmers are no doubt useful, a broader base of talent might produce more for society than a frictionless Saturday night.<sup>4</sup>

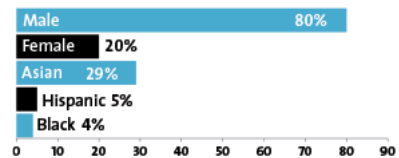
And there's evidence that diverse teams produce better products. A [study](http://www.ncwit.org/sites/default/files/legacy/pdf/PatentReport_wAppendix.pdf) ([http://www.ncwit.org/sites/default/files/legacy/pdf/PatentReport\\_wAppendix.pdf](http://www.ncwit.org/sites/default/files/legacy/pdf/PatentReport_wAppendix.pdf)) of 200,000 IT patents found that "patents invented by mixed-gender teams are cited [by other inventors] more often than patents invented by female-only or male-only" teams. The authors suggest one possibility for this finding may be "that gender diversity leads to more innovative research and discovery." (Similarly, [research](http://www.nber.org/papers/w19905) (<http://www.nber.org/papers/w19905>) papers across the sciences that are coauthored by racially diverse teams are more likely to be cited by other researchers than those of all-white teams.)

Fortunately, there's [evidence](http://www.citejournal.org/vol11/iss1/science/article1.cfm) (<http://www.citejournal.org/vol11/iss1/science/article1.cfm>) that girls exposed to very basic programming concepts early in life are more likely to major in computer science in college. That's why approaches like Margolis' ECS course, steeped in research on how to get and keep girls and other underrepresented minorities in computer science class, as well as groups like Black Girls Code, which offers affordable code boot camps to school-age girls in places like Detroit and Memphis, may prove appealing to the industry at large.

"Computer science innovation is changing our entire lives, from the professional to the personal, even our free time," Margolis says. "I want a whole diversity of people sitting at the design table, bringing different

## COMPUTER WORKFORCE

Percentage of today's software workforce that is...



Source: BLS

Percentage of computer science majors who were women:

37 percent in 1985

18 percent in 2009

18 percent in 2012

Department of Education

Mother Jones

<sup>3</sup> Six "ENIAC girls" did most of the programming, but until recently their work was all but forgotten. Male engineers worked on ENIAC's hardware, reflecting that until the 1950s, coding was considered clerical—even though it always involved higher math and applied logic. It was recast as a masculine pursuit as projects like Grace Hopper's UNIVAC demonstrated its promise.

<sup>4</sup> For example, Janet Emerson Bashen was the first black woman to receive a software patent, in 2006, for an app to better process Equal

sensibilities and values and experiences to this innovation. Asking, 'Is this good for this world? Not good for the world? What are the implications going to be?'"

*Employment Opportunity  
claims.*

We make kids learn about biology, literature, history, and geometry with the promise that navigating the wider world will be easier for their efforts. It'll be harder and harder not to include computing on that list. Decisions made by a narrow demographic of technocrat elites are already shaping their lives, from privacy and social currency, to career choices and how they spend their free time.



**Black Girls Code has introduced more than 1,500 girls to programming.** Black Girls Code

Margolis' program and others like it are a good start toward spreading computational literacy, but they need a tremendous amount of help to scale up to the point where it's not such a notable loss when a teacher like Kim Merino leaves the profession. What's needed to make that happen is for people who may never learn a lick of code themselves to help shape the tech revolution the old-fashioned way, through educational reform and funding for schools and volunteer literacy crusades. Otherwise, we're all doomed—well, most of us, anyway—to be stuck in the Dark Ages.

*Illustration by Charis Tsevis. Web production by Tasneem Raja.*

 Share on Facebook

(<http://facebook.com/sharer.php?u=http://www.motherjones.com/media/2014/06/computer-science-programming>

 Share on Twitter

([http://twitter.com/home?status=Is coding the new literacy? http://www.motherjones.com/media/2014/06/computer-science-programming](http://twitter.com/home?status=Is%20coding%20the%20new%20literacy?%20http://www.motherjones.com/media/2014/06/computer-science-programming)



**TASNEEM RAJA** (</authors/tasneem-raja/>) Interactive Editor

Tasneem Raja is MoJo's Interactive Editor. She specializes in web app production, interactive graphics, and user interface design. [RSS](#) ([RSS/AUTHORS/1315771](#))

[TWITTER \(HTTP://TWITTER.COM/TASNEEMRAJA\)](http://twitter.com/tasneemraja)

</authors/tasneem-raja/>

[tasneem-raja](#)

[raja](#)

#### IF YOU LIKED THIS, YOU MIGHT ALSO LIKE...

**[Jen Pahlka Wants to Reboot the US Government](/media/2012/05/code-for-america-jen-pahlka-interview/)** (</media/2012/05/code-for-america-jen-pahlka-interview/>)

The Code for America founder is taking her innovative tech agenda from the local level to the White House.

**[Not Everyone Needs to Learn Programming, But Every School Should Offer It](/media/2014/04/not-everyone-needs-to-learn-programming-every-school-should-offer-it/)** (</media/2014/04/not-everyone-needs-to-learn-programming-every-school-should-offer-it/>)

Surprisingly, plenty of schools don't have a single computer programming class.

**[Silicon Valley Firms Are Even Whiter and More Male Than You Thought](/media/2014/05/google-diversity-labor-gender-race-gap-workers-silicon-valley/)** (</media/2014/05/google-diversity-labor-gender-race-gap-workers-silicon-valley/>)

Our exclusive data shows that Google, which just released diversity numbers, lags further behind than other major tech

firms.

[Advertise on MotherJones.com \(/about/advertising/contact-form\)](#)

YOUR EMAIL

SIGN UP

## Featured Comment



**rmstallman** • 3 months ago

Like most discussions of democractizing programming skill, this one misses a crucial point:

Knowing how to program will get you nowhere when you use a program that users are not allowed to study or change.

When does this happen? If you are running a program that's not free/libre, whose source code is secret, you're blocked at the start. If the program is running in some company's server, a "cloud" gets in the way as you try to change it.

To make computing democratic, the users must control the software that does their computing!

See <http://gnu.org/philosophy/free...>  
and <http://gnu.org/philosophy/who-....>

Richard Stallman  
President, Free Software Foundation ([fsf.org](http://fsf.org))  
Internet Hall-of-Famer ([internethalloffame.org](http://internethalloffame.org))  
MacArthur Fellow

80 ^ | v • Share >

225 Comments **Mother Jones**

Login ▾

Sort by Oldest ▾

Share Favorite ★



Join the discussion...



**Nick Bell** • 3 months ago

Excellent article that further clarifies our need for kids (and especially girls) to have a love of science. The fact that many can take this love and turn it into a well paying career should make it an easier sell. Well I am in the pharmaceutical field, more and more of what we do has a computer basis to it. Even having a general knowledge and understanding is a HUGE leg up on your competition for jobs.

11 ^ | v • Reply • Share >



**Pummell** → Nick Bell • 3 months ago

Exactly. I would rather my daughters learn computer language (code) than a foreign language. Not that foreign languages are not important, but that code is universal, and my highschool French hasn't really been useful.

7 ^ | v • Reply • Share >



**iv sciguy** → Pummell • 3 months ago

My high school Spanish has been pretty useful. I travel a lot and knowing one of the Romantic languages has made it much easier to get around in much of Europe. I get the gist of most signs and menus and such no problem. Japan and Hong Kong would have been a lot tougher, except pretty much everyone under about 30 knows English.

7 ^ | v • Reply • Share ›



**Stacie** → iv sciguy • 2 months ago

We're talking about underprivileged kids here who most likely don't even have computers at home to use. You think they're going to be world travellers concerned about how to read menus at fancy cafes?

3 ^ | v • Reply • Share ›



**iv sciguy** → Stacie • 2 months ago

Hey, I grew up pretty poor and used to practice my Spanish with the workers in the kitchen of the Chinese restaurant I worked in as a teenager. I also didn't have a home computer until my dad's work gave him one they were just going to throw away when I was in Junior High. It was old DOS computer. I had some fun keeping it running and learning how to use it. Luckily I did really well in school and got through college and now have a great job. I actually traveled to both Europe and Asia as part of my job as an engineer. I then went back to France on vacation.

5 ^ | v • Reply • Share ›



**Robyn Ryan** → Stacie • 2 months ago

If Americans can't keep up with Japan and Hong Kong, then we have cheated our children out of the education they deserve. And they will spend their days serving those people who were educated.

1 ^ | v • Reply • Share ›



**RobertSF** → Stacie • 2 months ago

No, but you think they'll grow up and get jobs where knowing how to code or even thinking like a coder is useful? Nope, not in the United States of Inequality. The best that the majority of underprivileged kids have to look forward to in life is a job in fast food or retail. Sadly, most will instead be sent to prison.

1 ^ | v • Reply • Share ›



**2skeptical** → iv sciguy • 2 months ago

Ivy League Science Guy: How quaint. Did you enjoy your post-graduation European jaunt as a booby prize from your parents before taking an unpaid internship? Get a grip...

^ | v • Reply • Share ›



**iv sciguy** → 2skeptical • 2 months ago

Not what the Iv stands for. I actually grew up pretty poor, got a full scholarship to a state school. Then I got an awesome job. Went to those places for while traveling as part of my job.

2 ^ | v • Reply • Share ›



**Panagiotis Atmatzidis** → Pummell • 3 months ago

You might. I can write computer language (code) but I'm not sure that I'd exchange that for my English or Italian. Actually, I wouldn't.

1 ^ | v • Reply • Share ›



**Lord Byng** → Pummell • 3 months ago

Universal??? Universal??? Try Assembler, Fortran, Turbo Pascal, Clipper, Dbase 3, C, C++, Java, Visual Basic, C#, F#, Smalltalk, PHP, Asp Classic, **Asp.net**, Actionscript, Transact-SQL, Javascript, and JQuery. Those are just SOME of the languages I've made my living with over my time as a programmer.

In contrast to three-quarters of the above, French hasn't changed a lot since I was in high school. It hasn't become utterly useless and obsolete like most of them.

7 ^ | v • Reply • Share ›



**Stacie** → Lord Byng • 2 months ago

The point of this article was to teach computer literacy, not specific languages. Thinking like a programmer and being an expert in one specific syntax are two different things. This article is focusing on the former

6 ^ | v • Reply • Share ›



**Kayla** → Lord Byng • 2 months ago

Half the technologies you listed aren't even programming languages. Get off your high horse.

2 ^ | v • Reply • Share ›



**Donna Bradley** → Kayla • 2 months ago

tl bet you haven't heard of half of them. Your ignorance is astonishing.

^ | v • Reply • Share ›



**Kayla** → Donna Bradley • 2 months ago

I ol right Because ASP Classic and ASP NET totally don't fall under either the C# or VB languages (also listed) And

...at night, because ASP Classic and ASP.NET totally don't run under either the C# or VB languages (also listed). And JQuery isn't JavaScript at all. But for the sake of making an impressive list demonstrating how hard it is to be a programmer, let's just list every possible almost-language to make ourselves sound better. Sounds like the resume of a recent college graduate trying to impress the interviewer with the fact that he knows 15 languages, each of which he studied for a week.

4 ^ | v • Reply • Share >



**Kostas Karolemeas (allcancode)** → Pummell • 3 months ago

I guess choosing a foreign language over a computer one is a little bit exaggerated just to stress your point, but I agree that computer literacy is at least of equal importance. Computers are the machines that empower our minds in the Knowledge Revolution. Producing more developers is not the solution. We need to teach everyone how to solve computational problems without a developer. Developers are needed to build the platforms on which those solutions will be expressed.

1 ^ | v • Reply • Share >



**Nikolaus Phillips** → Pummell • 10 days ago

For everybody else who has replied, arguing about whether or not a) computer coding is universal, or b) preferable to a foreign language, I think you've all failed to understand this guy's comment - which is completely on-point.

When you learn another spoken language, it's useful in the context of a location or demographic where it's prevalent. I.E., I know some Japanese, but living on the east coast of the U.S. it doesn't do much for me. Knowing Spanish in many parts of the U.S. would be very useful, but knowing Spanish in Iceland probably isn't worth anything. On the other hand, HTML works the same everywhere it executes, and javascript I write on my computer this afternoon will execute on anybody's computer, anywhere in the world, whenever they load a page containing it. People in Finland or South Korea learning javascript will learn \*the same thing\*. It's universal. Moreover, even between the myriad computing languages out there, logical structures are universal. Don't tell me that learning variable scoping, object use, condition statements, loops, arrays, and error checking doesn't carry over from one syntax to the next - just to name a few examples. Again, these transcend geopolitical boundaries, and you can use them from your living room to touch anywhere in the world, right this minute. And everybody uses the same constructs, in the same general context, to adapt to the same changes happening all over the place every day.

As for their worth comparative to a second language, unless you plan on living or working in a foreign country, computer literacy is (by an overwhelming margin) more relevant to your life. Even if you are living/working in a foreign country, in many cases it'd still be a close call. Most of us are working with and/or carrying around 2 to 5 I.P. addresses each, destined to explode exponentially in the near future of ipv6, in the "internet of things." You might not care what that means, or care about

[see more](#)

1 ^ | v • Reply • Share >



This comment was deleted.



**iv sciguy** → IT 2 IT • 3 months ago

The first commercial computer system was actually used by American Airlines to track reservations and maintenance. Before that they were mainly used for code breaking and to test mathematical theories. What eugenists are you specifically talking about.

3 ^ | v • Reply • Share >



**mercedeslackey** → ivsciguy • 3 months ago

Actually not true. The first commercial computer systems were used by oil companies, first in their accounting departments, and then to help automate the refineries. My father worked on one of them, at Sinclair; with half an electrical engineering degree and all of an accounting degree they figured he was a good fit. This was in the very early 1950s, a good 5-10 years before Sabre was in operation (Sabre went online in 1960). Incidentally, I went on to become a programmer for Sabre in 1982.

2 ^ | v • Reply • Share >



**margaret diamond** → mercedeslackey • 2 months ago

I worked on Hollerith in the 1940's at War Office in London. Hollerith was the British equivalent of IBM and they were inter-changeable except for a couple of letters/numbers. The machines were huge and the programming we did was pretty simple. Our task was the Roll of Honor for those killed in WW2. I'm now 87 years old and have lots of time to spend on my Mac.

6 ^ | v • Reply • Share >



**mercedeslackey** → margaret diamond • 2 months ago

Holy moly! You're one of the Founding Mothers! Not to mention that you worked in the *War Office in London!*

It was pegboard programming, right? (actually sticking wired pegs into boards to create temporary connections?) Did you ever have the chance to meet Grace Hopper?


Someone should interview you to write up that story of Hollerith and the first days of programming during the war.

3 ^ | v • Reply • Share >




**margaret diamond** → mercedeslackey • 2 months ago

My first computer that worked better than an OO system and similar to the IBM cards of that time. It was coded


 No it was a keypunch that punched holes on an 80 column card similar to the IBM cards of that time. Info was coded into specific columns and then later cards were sorted in order to retrieve whatever info one needed. Then after sorting the cards they were tabulated on a huge noisy machine. After that the info was printed. I'll try and find a pic of the Hollerith tabulator taken in New Zealand as they used those machines there also.

4 ^ | v • Reply • Share >

 **mercedeslackey** → [margaret diamond](#) • 2 months ago


My father worked on the earliest models that literally did wired programming. You had to move the conductive pegs to different holes in order to change the programming. I worked on IBM keypunch card machines before they upgraded to programming on a "dumb" terminal, your cards had better be in the right sequence to compile, and woe betide you if you mis-punched a card. Kids these days! They have no idea how easy they have it! \*Shakes cane.\*

4 ^ | v • Reply • Share >

 **margaret diamond** → [mercedeslackey](#) • 2 months ago


We had verifiers who checked the keypunch operators work. In later years ...1978 on I taught elementary school and we had Atari computers in the classroom. The 'NEW' math had been introduced and this included teaching different number systems. The kids learned the binary system ...used in computer language... and were quite adept. The reason it was easier for 10 year olds to understand was that they hadn't used the decimal system as long as their parents. They learned how to add and subtract in base 2 ....it was amazing. Then they learned programming in 'Pilot' language which was easier learn than Basic Language for 10 year olds. This studying of 'Pilot' graphics programming could be transferred to other disciplines too. Those were very good teaching years as we were learning ....teachers and students ....as we went along.

4 ^ | v • Reply • Share >

 **margaret diamond** → [margaret diamond](#) • 2 months ago


Another learning program for children at that time was 'Logo' That programming and another one is still around. <http://scratch.mit.edu/>

2 ^ | v • Reply • Share >

 **Howard Cherniack** → [mercedeslackey](#) • 2 months ago

After my unsuccessful attempt to get my "Geniac" pegboard computer to work, my first computer, in 1960, was an IBM 610 (<http://www.columbia.edu/cu/com....>). It was about the size of a double-pedestal desk (perhaps the first personal computer); input was on a keyboard-sized console with a few switches and a small round CRT that could display the contents of any of its 84 "registers" (i.e., memory locations); output was a pre-Selectric IBM electric typewriter. Programming was either on a punched paper tape or a plugboard (the programming medium of choice for IBM's line of "tab" (Hollerith-card) equipment, like the 407 Accounting Machine). I remember spending a lot of time trying to set up a Newton's Method algorithm for solving polynomial equations.


1 ^ | v • Reply • Share >

 **mercedeslackey** → [Howard Cherniack](#) • 2 months ago

Were these your own personal computers or were they the property of a university?

Parenthetically, I think someone needs to start doing interviews with all the early programmers, even the "humdrum" ones like my dad, and get them archived at the National Library. There is a lot of early history here that should be saved.

1 ^ | v • Reply • Share >


 **Howard Cherniack** → [mercedeslackey](#) • 2 months ago

The \$15.00 pegboard Geniac was indeed mine, but the 610 belonged to Carleton College, where I spent much of my first semester in the computer lab trying to dodge freshman hazing.

For better or worse, I transferred my attentions to liberal arts, and didn't even work with the IBM 1620 (a classic) when it replaced the 610. It was only in graduate school in Berkeley that I started getting back into computers as a sideline to my work in City and Regional Planning, first with an IBM 7040/7094 and then with a CDC 6400, both mainframes. A friend of mine who stuck with computers all the way through made many innovations and is probably a zillionaire now. I may not have been as smart as he was, but if I had stuck with computers (my eventual career) from the beginning, maybe I could have been a contender...

As a sidebar, let me mention that in those days computers weren't really a career, but a calling--how great of the university of give us such wonderful toys. And, in the universities at least, it seemed to me that there were as many women as men who felt the calling. In fact, there was one hot-shot female programmer at Berkeley who was reputed to have said that women were inherently and genetically better programmers than men.

2 ^ | v • Reply • Share >

 **mercedeslackey** → [Howard Cherniack](#) • 2 months ago

Early programming seemed to have gotten stuck halfway between EE and Accounting, which was how my father ended up in it (he had half an EE degree and a full accounting degree, which was why Sinclair snapped him up for their early Univac). And yes, it was one of those things that had to be a calling. Early programmers really had to be able to think in flow-charts. And then have the patience to wade through all of the compile errors to finally arrive at a finished product that actually did what you wanted it to in the teeniest amount of memory.



I'm dead certain that it wasn't even possible to get a degree in programming or computer science until about the mid-60s. I'm not sure how other universities did it, but at Purdue all the computer science classes were lumped under the EE department and you got an EE degree. I remember when I went to Purdue, the EE building was always open all night long for those poor souls trying to slog their way through compilation errors.

^ | v • Reply • Share >



**Howard Cherniack** → mercedeslackey • 2 months ago

Yes. At Carleton, the 610 was under the aegis of the Math department; at Berkeley, to \*study\* computers, you went to the EE department, but the Computation Center was seen as a service to the academic community, and seemed (as far as I could tell) to operate pretty independently.

I sure remember those nights. During the daytime, maybe eight hours was par for card input to print output, but at night it was maybe one or two hours.

I still have my IBM flow-chart template, plus the "green card" and its successors, for reading core dumps. Fun times.

1 ^ | v • Reply • Share >



**mercedeslackey** → Howard Cherniack • 2 months ago

Oh god....core dumps. Why is it the stupid thing *always* crashed at 3 AM and never during normal daylight hours?

At one time I could program in machine language....and read it. But then, I was an ASM programmer, and that's not that far off from machine code.

My brother still can code in machine language. He ended up with one of those Purdue CompSci/EE degrees.

We really need someone to start collecting stories about the truly early days of programming. Things like what it was like to work with Grace Hopper, or on those early DoD computers. Maybe verify that the term "bug" really did come from a cockroach getting shorted out on a pegboard.

^ | v • Reply • Share >



**Howard Cherniack** → mercedeslackey • 2 months ago

Perhaps then, you remember the infamous 0C4 error code on OS/360. And I even had a copy of "The Programmer's Guide to Debug," which was the only place you could find out what the SVC codes were (as I recall).

When Berkeley switched its heavy metal from IBM to CDC, we had a chance to work on what later became known as a Reduced Instruction Set Computer. It has so few op codes that you really didn't need the equivalent of a green card to read a dump. It didn't even have a fixed-point multiply, but it could convert a couple of integers to floating-point, multiply them, and then convert them back to fixed-point faster than a 360 could do a fixed-point multiply.

^ | v • Reply • Share >



**mercedeslackey** → Howard Cherniack • 2 months ago

0C4? You are resurrecting my nightmares! *Always, always* at 3 AM.....

When I was actually a working programmer I was working in ASM in PARS/SABRE (that's the airline res system and a strange and bizarre OS it is) for an airline, and, thank heavens, PARS does not permit calculations in anything other than a working block of memory that is passed along through 100% re-enterable modules. That much, at least, prevented a whole lot of 0C4s. I never had to handle the batch-handling COBOL programs. Most of our 0C4s occurred when something else decided to walk all over the memory and obliterated our code and/or storage blocks.

^ | v • Reply • Share >



**Howard Cherniack** → mercedeslackey • 2 months ago

Very famous system, SABRE, along with its uniformed counterpart SAGE. I see that it first ran on a 7090, but couldn't have 0C4s until it was ported to 360--in ASM, that must have been quite a job in itself. Yup, walking over memory will do it, for sure.

Let me digress (so to speak) a bit for a comment on the original article. I have to say that for most of my career, "coders" were considered the bottom of the programming chain. In big rigid shops, the system analysts would design the outline of a system, the programmers turn this into sort of a draft or pseudo-code model, and then coders would translate that model into the language du jour. In smaller shops, a programmer would create a system design and then turn that into code. One watchword I remember is, "First thing, put down your pencil and think about the problem." This article is a bit more nuanced than, for instance, the HBO show "Halt and Catch Fire" on this point, but most of the programmers I knew would resent being referred to as "just" "coders."

I presume that you are the author? I'm afraid that I haven't read a great many of your books, but those I have read I have enjoyed a lot.

^ | v • Reply • Share >



**mercedeslackey** → Howard Cherniack • 2 months ago

My dad became a system analyst as soon as they had such things; when he started, it was "all hands on deck," as it were. By the time I entered the profession, the "systems analysts" were the ones designing the new applications; I was in maintenance, and it was "all hands on deck" in there too.



But yes, most of the programmers I knew back when I was still in the trenches would hate being called "coders." Kids these days! (grin)(shakes cane)

In fact, yes I am the author, and thank you! (blush)

^ | v • Reply • Share >



**dgrb** → margaret diamond • 2 months ago

Hollerith wasn't the "British equivalent of IBM". Herman Hollerith was an American engineer who designed his punch card tabulator (punched cards were designed by Jacquard for his loom attachment and later used by Babbage) for the US Census.

Hollerith's company was one of four which merged in 1911 to form the Computing Tabulating Recording Company, which was renamed in 1924, International Business Machines, although the punch card machines continued to be called Hollerith machines for several decades.

^ | v • Reply • Share >



**dgrb** → mercedeslackey • 2 months ago

The very first computer built to be sold rather than built on commission was the Ferranti Mark 1, a commercially-engineered version of the Manchester Mark 1 computer, which, as the Small Scale Experimental Machine (aka "Baby") ran the world's first computer program in June 1948.

The Ferranti machine was on the market several months before the Univac. This was in early 1951. The first model went to Manchester University, the second to the University of Toronto, where it was used in the design of the St. Lawrence Seaway. William Kahan, co-designer of the IEEE 754/854 floating point standard first programmed on the UofT machine.

^ | v • Reply • Share >



**mercedeslackey** → dgrb • 2 months ago

Excellent, thank you for correcting me. I had always thought the first commercial system was the Univac (which my father worked on). Have you ever seen one of these first "small scale" machines? They practically fill a building!

^ | v • Reply • Share >



**dgrb** → mercedeslackey • 2 months ago

Well, US authors have always been loathe to admit that it wasn't invented there...:-)

The Baby filled an entire room and late extensions filled other rooms, not necessarily on the same floor! There is a description of programming the Manchester machine by Alan Turing's assistant, the deliciously-named Cicely Popplewell, in which she mentions going down stairs to mount a tape then going back up to the main machine, hoping the tape drive was still functioning. Traffic in the street could cause errors.

The earliest machine I ever saw/used was the IBM 7090 belonging to Imperial College in London. My school was one of three in a pilot project to learn to program it (this was early 1968). Put me off FORTRAN for life.

^ | v • Reply • Share >



**mercedeslackey** → dgrb • 2 months ago

The original Univac at Sinclair Oil (in East Gary Indiana) was just about as temperamental. I remember Dad coming home one day early because lightning had struck nearby...not on the building, just somewhere nearby...and shorted out a key component called the "supervisor."

^ | v • Reply • Share >



**Bronx2216** → mercedeslackey • 2 months ago

OMG - Mercedes Lackey AND Peg Diamond. I'm swooning (yes, I'm a geek)...and the conversation is astounding.

1 ^ | v • Reply • Share >



**mercedeslackey** → Bronx2216 • 2 months ago

just got a couple new posts to the thread, FYI

1 ^ | v • Reply • Share >



**Kir (Politicoïd)** • 3 months ago

Okay, I get the importance of learning how to code. However, as for what is most important in education and what schools need to do... Sorry. No.

Schools need to nurture the love of learning and need to teach, first and foremost, the skills needed to learn. You don't need school to learn, once you have all the basic tools and the desire to do so. However, the school system, as it exists today, does the exact opposite of what it needs to do. It tries to drill in specific information and it cripples the love of learning. It creates a bunch of semi-obedient/semi-mindless test takers who become good little industrial workers.

18 ^ | v • Reply • Share >



**Geldscheisser** → Kir (Politicoïd) • 3 months ago

That is because it is modeled on the Prussian school system that was designed to create a bunch of semi-obedient/semi-mindless

...that created it is modeled on the Prussian school system that was designed to create a nation of both obedient and obedient soldiers who obeyed their commanders.

5 ^ | v • Reply • Share ›



**Kir (Politicoid)** → Geldscheisser • 3 months ago

You're are correct, but I would like to add more to that.

Even before that it was modeled after the church style education system which was used to indoctrinate people into the faith. Prussians picked it up and the United States picked it up during the industrial revolution so that it could create good little workers.

6 ^ | v • Reply • Share ›



**Peaches** → Kir (Politicoid) • 3 months ago

I'd just like to make a small point here...teachers don't like what the testing regimen has done to our schools either. As an instructor of elementary assessment for teachers, I see so much outrage and disgust among new teachers-in-training focused on the supposed "drilling in of specific information", aka "teaching to the test".

That tactic does not actually improve test scores, which have become sophisticated enough to check for comprehension and application. Outright memorization and regurgitation does not work, especially with the new Common Core testing models. Students who have successfully attained a love of learning and a good holistic preparation from their teachers tend to do well on the tests.

Good education is happening in many of our schools despite many shortsighted initiatives, and it could be so much better if we tried to nurture the whole child rather than trying to target the needs of industry from the cradle without a decent vocational schooling track.

5 ^ | v • Reply • Share ›



**Geldscheisser** → Kir (Politicoid) • 3 months ago

I think for all but the disadvantaged and undisciplined, some form of loosely organized home schooling, done on the internet, would provide a much better education. Since I don't have kids, I don't know what's already out there. Unfortunately working parents would balk at this too, because school is their babysitter. Truly education needs to be (and is being) reinvented. I had a Slovenian exchange student stay with me and do his senior year of H.S. here. Despite barely speaking English he got straight A's and couldn't understand how we could be so rich yet so stupid.

4 ^ | v • Reply • Share ›



**margaret diamond** → Kir (Politicoid) • 2 months ago

"the United States picked it up during the industrial revolution so that it could create good little workers."

The Industrial Revolution happened in England and that was where the schools suddenly became the place for the peons to learn 'just enough' but not too much so that they would make good workers.

^ | v • Reply • Share ›



**Kir (Politicoid)** → margaret diamond • 2 months ago

The industrial revolution did not occur in only a single place. however, the point is that our education system is not based an approach that provides a solid education. It is based on creating obedient worshipers or workers.

^ | v • Reply • Share ›

Load more comments

ALSO ON MOTHER JONES

**Idaho Professor Accidentally Shoots Himself While Teaching Class** 806 comments

**How a Bipartisan Education Reform Effort Became the Biggest Conservative Bogeyman Since ...** 100 comments

**This Map Shows Why People Are Freaking Out About Ebola's Arrival in Senegal** 42 comments

AROUND THE WEB

**Please Don't Retire At 62. Here's Why.** The Motley Fool

**How To Invest \$100, \$1,000, Or \$10,000** Business Insider

**Heart Attack: How Your Body Warns You Days Before** Newsmax Health

**Controversial "Skinny Pill" Sweeps the Nation** HealthyLifeStyle

WHAT'S THIS?

✉ Subscribe

Ⓓ Add Disqus to your site

MoJo Troll Patrol encourages readers to sign in with Facebook, Twitter, Google, Yahoo, Disqus, or OpenID to comment. Please [read our comment policy](#) [/about/interact-engage/comment-policy](#) before posting.



Beginner's Guide to Git  
A visual tutorial to Git workflows & commands.

AdChoices

Learn more

## PHOTO ESSAYS • SLIDESHOWS



### **10 Stunning Photos That Illuminate Unseen Stories From Around the Globe**

This year's Magnum Foundation Emergency Fund grant winners explore underreported issues and looming crises from Alabama to Venezuela.



### **This Is What a Farmer Looks Like**

FarmHer documents the badass women who grow our food.



### **The World's Largest Church Is in the Middle of an African Coconut Plantation**

Côte d'Ivoire's longtime dictator spent millions on this massive vanity project.



### **Photo Essay: Chasing Meth in Laurel County, Kentucky**

Stacy Krantz's intimate look at methamphetamine addicts and the cops trying to nab them.

[SEE ALL](#)