TIME

Fifty Years of BASIC, the Programming Language That Made Computers Personal

Early in BASIC's history, its creators, John Kemeny (left) and Thomas Kurtz (center) go over a program with a Dartmouth student
Adrian N. Bouchard / Dartmouth University

April 29, 2014

Knowing how to program a computer is good for you, and it’s a shame more people don’t learn to do it.

For years now, that's been a hugely popular stance. It's led to educational initiatives as effortless sounding as the Hour of Code (offered by Code.org) and as obviously ambitious as Code Year (spearheaded by Codecademy).

Even President Obama has chimed in. Last December, he issued a YouTube video in which he urged young people to take up programming, declaring that "learning these skills isn't just important for your future, it's important for our country's future."

I find the "everybody should learn to code" movement laudable. And yet it also leaves me wistful, even melancholy. Once upon a time, knowing how to use a computer was virtually synonymous with knowing how to program one. And the thing that made it possible was a programming language called BASIC.

Invented by John G. Kemeny and Thomas E. Kurtz of Dartmouth College in Hanover, New Hampshire, BASIC was first successfully used to run programs on the school's General Electric computer system 50 years ago this week–at 4 a.m. on May 1, 1964, to be precise.

The two math professors deeply believed that computer literacy would be essential in the years to come, and designed the language–its name stood for "Beginner's All-Purpose Symbolic Instruction Code"–to be as approachable as possible. It worked: at first at Dartmouth, then at other schools.

In the 1970s and early 1980s, when home computers came along, BASIC did as much as anything else to make them useful. Especially the multiple versions of the language produced by a small company named Microsoft. That's when I was introduced to the language; when I was in high school, I was more proficient in it than I was in written English, because it mattered more to me. (I happen to have been born less than a month before BASIC was, which may or may not have anything to do with my affinity for it.)

BASIC wasn't designed to change the world. "We were thinking only of Dartmouth," says Kurtz, its surviving co-creator. (Kemeny died in 1992.) "We

needed a language that could be ‘taught’ to virtually all students (and faculty) without their having to take a course.”

Their brainchild quickly became the standard way that people everywhere learned to program computers, and remained so for many years. But thinking of its invention as a major moment only in the history of computer languages dramatically understates its significance.

In the mid-1960s, using a computer was generally like playing chess by mail: You used a keypunch to enter a program on cards, turned them over to a trained operator and then waited for a printout of the results, which might not arrive until the next day. BASIC and the platform it ran on, the Dartmouth Time Sharing System, both sped up the process and demystified it. You told the computer to do something by typing words and math statements, and it did it, right away.

"We were thinking only of Dartmouth."

Today, we expect computers–and phones, and tablets and an array of other intelligent devices–to respond to our instructions and requests as fast as we can make them. In many ways, that era of instant gratification began with what Kemeny and Kurtz created. Moreover, their work reached the public long before the equally vital breakthroughs of such 1960s pioneers as Douglas Engelbart, inventor of the mouse and other concepts still with us in modern user interfaces.

You might assume that a programming language whose primary purpose was to help almost anybody become computer-literate would be uncontroversial—maybe even universally beloved. You’d be wrong. BASIC always had its critics among serious computer science types, who accused it of promoting bad habits. Even its creators became disgruntled with the variations on their original idea that proliferated in the 1970s and 1980s.

And eventually, BASIC went away, at least as a staple of computing in homes and schools. Nobody conspired to get rid of it; no one factor explains its gradual disappearance from the scene. But some of us miss it terribly.

When it comes to technology, I don't feel like a grumpy old man. Nearly always, I believe that the best of times is now. But I don't mind saying this: The world was a better place when almost everybody who used PCs at least dabbled in BASIC.

BASIC Beginnings

Sooner or later, it was inevitable that someone would come up with a programming language aimed at beginners. But BASIC as it came to be was profoundly influenced by the fact that it was created at a liberal arts college with a forward-thinking mathematics program. Dartmouth became that place largely because of the vision of its math department chairman, John Kemeny.

Born in Budapest in 1926 and Jewish, Kemeny came to the United States in 1940 along with the rest of his family to flee the Nazis. He attended Princeton, where he took a year off to contribute to the Manhattan Project and was inspired by a lecture about computers by the pioneering mathematician and physicist John von Neumann.

Kemeny worked as Albert Einstein's mathematical assistant before arriving at Dartmouth as a professor in 1953, where he was named chairman of the mathematics department two years later at the age of 29. He became known for his inventive approach to the teaching of math: When the Alfred P. Sloan Foundation gave the school a $500,000 grant to build a new home for the department in 1959, TIME noted the news and said it was mostly due to Kemeny's reputation.

The thinking that led to the creation of BASIC sprung from "a general belief on Kemeny's part that liberal arts education was important, and should include some serious and significant mathematics–but math not disconnected from the

general goals of liberal arts education," says Dan Rockmore, the current chairman of Dartmouth's math department and one of the producers of a new documentary on BASIC's birth. (It's premiering at Dartmouth's celebration of BASIC's 50th anniversary this Wednesday.)

"Our vision was that every student on campus should have access to a computer."

In the early 1960s, average citizens–even those who happened to be students at Ivy League schools with computing centers–had never encountered a computer in person. The machines were kept "behind locked doors, where only guys–and, once in a while, a woman–in white coats were able to access them," Rockmore says.

Kemeny believed that these electronic brains would play an increasingly important role in everyday life, and that everyone at Dartmouth should be introduced to them. "Our vision was that every student on campus should have access to a computer, and any faculty member should be able to use a computer in the classroom whenever appropriate," he said in a 1991 video interview. "It was as simple as that."

Of course, Dartmouth couldn't give a computer to every student and faculty member: Computers were a pricey shared resource, normally capable of performing only one task at a time. That's why you typically handed your program over on punch cards and waited your turn.

Tom Kurtz, who had joined Dartmouth's math department in 1956, proposed using a relatively new concept called time-sharing. It would divvy up one system's processing power to serve multiple people at a time. With what came to be known as the Dartmouth Time-Sharing System, or DTSS, a user sitting at a terminal would be able to compose programs and run them immediately.

"If you're trying to get a student interested in the idea of computing, you need some immediacy in the turnaround," says Rockmore. "You don't want to ship a 10-line program off to a computer center before you know if you've got it right."

But what sort of programs? In the past, Kemeny and Kurtz had made two unsuccessful stabs at creating computer languages for beginners: Darsimco (Dartmouth Simplified Code) and DOPE (Dartmouth Oversimplified Programming Experiment). But this time they considered modifying an existing language.

"I tried, briefly, to develop simple subsets of Fortran and ALGOL, but found quickly that such could not be done," Kurtz says. Even the most common of tasks could be tricky in Fortran, which had an "almost impossible-to-memorize convention for specifying a loop: '**DO 100, I = 1, 10, 2**'. Is it '**1, 10, 2**' or '**1, 2, 10**', and is the comma after the line number required or not?"

"Fortran and ALGOL were too complex," says John McGeachie, who, as a Dartmouth undergraduate, was the co-author of the DTSS software. "Anything that required days and days of training would have defeated the purpose. It certainly would have curtailed its widespread popularity."

So Kemeny and Kurtz decided to create something so straightforward that it almost didn't involve memorization at all. "We wanted the syntax of the language to consist of common words, and to have those words have a more-or-less obvious meaning," says Kurtz. "It is a slight stretch, but isn't it simpler to use **HELLO** and **GOODBYE** in place of **LOGON** and **LOGOFF**?"

"If you were writing a very simple program, you'd get your answer in a second or so."

BASIC was primarily Kemeny's idea, and he wrote the first version himself. Starting in September 1963, he and Kurtz began the overarching effort to get

the language and the DTSS up and running. They led a team of a dozen undergraduate students–young men who were still in the process of learning about computers themselves. (Dartmouth was a male-only institution at the time: Kemeny himself took it co-ed in 1972 as president of the college, a position he held from 1970-1981.)

“We used to work all night and then go to sleep,” remembers McGeachie. “Kemeny would work with us, and then go teach math to undergraduates.”

A $300,000 grant from the National Science Foundation helped fund the undertaking, which required not one but two powerful computers, both from General Electric. A GE-225 mainframe (quickly replaced with a faster GE-235) did the heavy lifting of performing floating-point math, while a smaller Datanet-30 coordinated communications with Teletype machines–essentially glorified typewriters–which students would use to do their programming.

“We were not working under much constraints,” Kurtz says. “We had 16K of 20-bit words to work with.” Though a rounding error by today’s standards, that was enough memory to write a capable version of BASIC: Years later, when others adapted the language for PCs, they sometimes had to cram it into as little as 3K of 8-bit memory, resulting in cut-down, ungainly implementations that Kemeny and Kurtz disowned.

Unlike many BASICs to come, Dartmouth BASIC was a compiler, which meant that it converted your entire program in one fell swoop into machine code that the computer could understand, rather than line by line every time you ran the program. It performed that task rapidly, especially by the leisurely standards of 1960s computing: “If you were writing a very simple program, you’d get your answer in a second or so,” McGeachie says. “It might take longer to print it out, because the Teletypes could only do 10 characters a second.”

The historic moment at Dartmouth on May 1, 1964 at 4 a.m. was actually two historic moments. Not one brief BASIC program but two or three of them–accounts vary–ran simultaneously, proving both that BASIC worked and that

the Dartmouth Time-Sharing System was capable of dealing with more than one user at a time.

In June 1964, they became generally available to Dartmouth students, initially on 11 Teletype machines. The first version of BASIC had 14 commands, all with straightforward names and syntax that made sense:

- **PRINT** output text and numbers to the Teletype (and, later on, displayed it on the screens of time-sharing terminals and PCs);
- **LET** told the computer to perform calculations and assign the result to a variable, in statements such as **LET C = (A*2.5)+B;**
- **IF** and **THEN** let the program determine if a statement was true, vital for anything involving decision-making;
- **FOR** and **NEXT** let a program run in loops;
- **GOTO** let a program branch to another numbered line within itself;
- **END**, which was required in Dartmouth BASIC, told the computer that it had reached the program's conclusion.

Then there was **INPUT**, a command that let a BASIC program accept alphanumeric characters typed in by a user. It wasn't among the initial 14, arriving only in the third revision of the language in 1966. But when it did, it made it possible to write far more interactive programs. Without **INPUT**, BASIC was mostly for solving math problems and doing simple simulations; with it, the language could do almost anything. Including play games, which many people came to consider as the language's defining purpose.

You could write a fairly sophisticated program in Dartmouth BASIC. (An early manual stated the maximum program length as "about two feet of teletype paper.") But you could also make the computer do something interesting and useful with just a few lines of simple code, shortly after you'd encountered the language for the first time. That was the whole point.

It mattered to Kemeny and Kurtz that access to BASIC and the DTSS be as open as possible. "Any student can enter the Library, browse among the books or take some back to his room. No one asks him why he wants the book, and he

does not need anyone's permission," Kemeny wrote in a brochure for the college's new computer center, which opened in 1966. "Similarly, any student may walk into the Kiewit Computation Center, sit down at a console, and use the time-sharing system. No one will ask if he is solving a serious research problem, doing his homework the easy way, playing a game of football, or writing a letter to his girlfriend."

What Kemeny was describing in the Kiewit brochure was personal computing. It's just that the term hadn't been invented yet. Even the concept was still audacious.

Dartmouth BASIC did everything that Kemeny and Kurtz hoped it would, and more. In a triumphant 1967 report, they said that by the end of that academic year, 2000 Dartmouth students–representing 80 percent of the three incoming freshman classes who had arrived since BASIC's invention–would have learned about computers by writing and debugging their own programs. Many continued to do so after completing the BASIC classwork that was a mandatory part of the school's math program. Forty percent of faculty members–not just math and science teachers–also used the system.

"Anyone who tries to convince a Dartmouth undergraduate either that computers are to be feared or that they are of little use, will be met with well-founded scorn," the report said. "The Dartmouth student knows better–and knows it from personal experience."

Dartmouth provided access to the DTSS over telephone lines to other East Coast schools, including Harvard and Princeton, as well as to some high schools. It also helped other institutions implement time-sharing systems and BASIC, while General Electric commercialized the DTSS and Dartmouth BASIC and sold them to business customers. Other computer companies such as Digital Equipment Corporation and HP introduced their own BASICs.

Dartmouth's effort to democratize computing was, in short, a huge success. "Qualitatively, I was right on the impact," Kemeny said in the 1991 interview. "Quantitatively, I vastly underestimated it. That is, it had impact on many,

many more courses than I thought, and the amount of impact was much greater–courses being totally changed because of the availability of computers. I also underestimated, of course, how far educational computing would spread all over the world."

Bashing BASIC

Not everybody was happy with the way the language put computing within reach of mere mortals. Its most articulate and vociferous opponent was Edsger Dijkstra (1930-2002), an influential computer scientist. "It is practically impossible to teach good programming to students that have had a prior exposure to BASIC," he groused in a 1975 essay titled "How Do We Tell Truths That Might Hurt?" "As potential programmers they are mentally mutilated beyond hope of regeneration."

Now, it's possible that Dijkstra was exaggerating for dramatic effect. BASIC wasn't his only *bête noire* among programming languages: He also spewed bile in the direction of FORTRAN (an "infantile disorder"), PL/1 ("fatal disease") and COBOL ("criminal offense").

Still, despite Dijkstra's foreboding attitude towards BASIC, countless programmers who started out with the language went on to have thriving careers. And the thing is, some of the characteristics that have given BASIC a bad reputation are precisely the same ones that made it so easy to learn.

For instance, BASIC offered **GOTO**, a command that let you jump from anywhere in your program to anywhere else in it—a practice that could result in messy "spaghetti code." (In 1968, Dijkstra devoted an entire essay to his contempt for the command, "Go To Statement Considered Harmful.") A thoughtful BASIC programmer could indeed compose fastidious code that didn't use **GOTO**. But insisting that liberal arts students obsess about tidy programming techniques from the get go was hardly a way to make computers less threatening. For them, **GOTO** was a godsend.

"It is practically impossible to teach good programming to students that have had a prior exposure to BASIC."

In its classic form, BASIC also made you organize your programs with line numbers—such as the **10** in **10 PRINT "HELLO"**—a convention that was arguably superfluous and eventually fell by the wayside. But line numbers helped emphasize the sequential nature of computer programs, which, regardless of the language in question, consist of a task broken down into steps.

In "How Do We Tell Truths That Might Hurt?," Dijkstra tips his hand by calling programming "one of the most difficult branches of applied mathematics" and suggesting that less talented mathematicians should not even bother with it. If that was his take in 1975, he couldn't simultaneously approve of BASIC. Either programming a computer was exceptionally hard and should be left to the experts, or it was something that should be democratized, as BASIC had already done. Not both.

Today, Kurtz is blunt about criticism of the language he co-created as being insufficiently serious or a dangerous way to begin learning about computer programming. "It's B.S.," he says.

"I'll go out on a limb and suggest the degrading of BASIC by the professionals was just a little bit of jealousy–after all, it took years for us to develop our skill; how is it that complete idiots can write programs with just a few hours of skill?"

BASIC may not have made sense to people like Edsger Dijkstra. That was O.K.—it wasn't meant for them. It made plenty of sense to newbies who simply wanted to teach computers to do useful things from almost the moment they started to learn about programming. And in 1975, as Dijkstra was accusing it of

mutilating minds, there were about to be far more of those people than ever before.

Enter the PC

By letting non-computer scientists use BASIC running on the DTSS, Kemeny, Kurtz and their collaborators had invented something that was arguably the first real form of personal computing. But it didn't yet involve personal computers. That revolution got jump-started a decade later, when a New Mexico model rocket company called MITS launched the Altair 8800, the $497 build-it-yourself microcomputer ($621 assembled) that launched the PC revolution.

It was huge news among the small number of people who could be called computer nerds at the time—people like Paul Allen, who was working as a programmer for Honeywell in Boston.

When he bought a copy of the January 1975 issue of *Popular Electronics* at the Out of Town newsstand in Harvard Square, with the Altair on the cover, he and an old friend—a Harvard sophomore named Bill Gates—got excited. Immediately, they knew they wanted to try to make the Altair run BASIC, a language they'd both learned in its original timeshared-via-Teletype form at the Lakeside School in Seattle.

Actually, Allen had been ruminating about the possibility of building his own BASIC even before he knew about the Altair. "There hadn't been attempts to write a full-blown programming language for a microprocessor," he explains. "But when the chips leading up to the 8080 processor became available, I realized we could write a program for it that would be powerful enough to run BASIC."

"I realized we could write a program for it that would be powerful enough to run BASIC."

Famously, they wrote the first draft of Altair BASIC without having access to an Altair, using a simulator they ran on a Digital Equipment PDP-10 minicomputer. "Monte Davidoff, who helped me and Bill write BASIC for the Altair, once said programming was like writing a novel," Allen says. "That's how we approached BASIC. We started with a plot of sorts to know the general approach we were taking. We broke the big ideas into sections, or chapters, and then we'd edit and re-edit and keep editing until we had preserved the big picture and fixed all the bugs. It was the hardest but most enjoyable work I've ever done."

Allen and Gates' BASIC partnership became a company, known at first as Micro-Soft. They licensed the language to MITS, which sold it for a hefty $500, marked down to $75 if you bought it bundled with Altair hardware. Or you could get it for free by pirating it—something that so many early microcomputer owners did that Gates wrote a legendary open letter bemoaning the rampant theft of Micro-Soft's intellectual property.

Setting a high price for BASIC and using it as an incentive to buy hardware "was a strategy that backfired in a big way," says David Bunnell, who was in charge of publications at MITS. Ed Roberts, MITS' president, "was just sort of small-minded in that way. He couldn't see the big picture."

Even so, BASIC being available on a microcomputer was a big deal. "There are two keys to the new computer revolution," declared an unsigned article–Bunnell says he almost certainly wrote it–announcing Altair BASIC in the April 1975 issue of MITS' *Computer Notes* newsletter. "One is computers must be inexpensive and the other is computers must be understandable. With the Altair 8800 and Altair BASIC, both of these criteria have been met."

That was true, but it was only the beginning of the story. The Altair and its earliest rivals catered to hobbyists who were comfortable flipping switches and wielding soldering guns. In 1977, a new wave of PCs arrived that were aimed at a far broader swath of consumers, including Apple's Apple II, Commodore's PET 2001 and Radio Shack's TRS-80. And one of the things that made them consumer PCs was that they came with BASIC.

The PET offered a variant of Microsoft BASIC from the start. Apple and Radio Shack started out with rudimentary variants of the language—Apple's was written by cofounder Steve Wozniak himself—before acquiring the rights to distribute Microsoft's version. For the next few years, nearly every microcomputer of any significance came with Microsoft BASIC, including models from companies such as Atari and Texas Instruments.

Tech pundits like to talk about *killer apps*–software so useful that you'd buy a computer just to run it. 1979's VisiCalc is often identified as the first such program. But before that, Microsoft BASIC itself was the PC's killer app. Lots of people bought a computer so they could learn how to program it.

There were other microcomputer BASICs besides Microsoft's. In the early days, the best-known of them was CBASIC, created by a naval officer named Gordon Eubanks. His version was particularly popular among people developing commercial programs–which, at the time, were as likely to be written in BASIC as in any other language.

"Microsoft's BASIC was much more fundamental than CBASIC," says Eubanks. "It appeared on every single computer. If you got your TRS-80, you could fire it up and write a little program to print 'HELLO.' And it worked. I was focused on a narrower thing, how to develop commercial applications...The end result is, Bill did a little better."

(Though he may not have reached Gates-ian levels of success, Eubanks did end up doing rather well for himself, eventually becoming the longtime CEO of Symantec, an enduringly important software company in its own right.)

Eubanks mentions a key reason for Microsoft BASIC's importance: It was unavoidable. When you turned on an early microcomputer such as the TRS-80, it dumped you directly into the language. You could load a program off tape cassette or floppy disk if you chose, but you could also simply begin typing a new one. These computers begged to be programmed in a way that their descendants do not.

Unlike both Dartmouth BASIC and CBASIC, Microsoft BASIC was an interpreted language, which meant that it converted each line of a BASIC program into machine code on the fly, every time you ran the program. That made it sluggish, especially if you didn't know any hacks for speeding up stuff like animated graphics.

But it also meant that anybody could examine the original program for anything written in Microsoft BASIC, including commercial software distributed on cassette or floppy disk. You could learn from it, tweak it or steal snippets of it for your own projects. It had many of the virtues of open-source software long before the concept had that name.

If you needed help learning BASIC's lingo, there was a near-infinite amount of reference material, all of which included programs you could type in and customize. Like the language itself, the works documenting BASIC tended to be rather informal by computer-science standards.

In 1972, for instance, Bob Albrecht of the People's Computer Company in Menlo Park, California wrote a primer with a wonderfully evocative title: *My Computer Likes Me When I Speak in BASIC*. You couldn't imagine a similar title being applied to a volume about FORTRAN (or, today, one about Objective-C or Java).

The single most influential book of the BASIC era was not a textbook–at least not officially. It was *101 BASIC Computer Games*, later known as *BASIC Computer Games* and edited, in both versions, by David H. Ahl.

The tome grew out of Ahl's work as educational marketing manager for Digital Equipment Corporation's PDP-8 line of minicomputers. Around 1971, he created BASIC versions of two games that had originally been written in DEC's FOCAL language, Hammurabi (in which the player governed ancient Sumeria) and Lunar Lander (exactly what it sounds like). "I wrote them from scratch more to demonstrate the machine than anything else," he says.

Their popularity led him to include BASIC games in *EDU*, a DEC newsletter he edited. Many of them were contributed by readers–especially high school students–and in 1973, DEC published an anthology, *101 BASIC Computer Games*.

The book quickly went into a second printing, for a total of 10,000 copies sold. "That was far more books than there were computers around, so people were buying three, four, five of them for each computer," Ahl remembers.

Shortly after Ahl left DEC in 1974, he started a magazine, *Creative Computing*, devoting a great deal of its space to BASIC. Then he acquired the rights to publish *101 BASIC Computer Games* under the title *BASIC Computer Games*.

As with BASIC itself, *BASIC Computer Games* predated the PC revolution but helped propel it forward. It got updated, was translated into six languages, inspired multiple sequels and became the first computer book to have sold a million copies. Like folk songs, its programs felt like part of a shared cultural heritage. They were passed around, mutating into multiple variants as they did so.

By the time I got my hands on the book–circa 1978 when my father brought home a TRS-80–I was aware that the games it contained were, well, basic. Because they had their roots in the Teletype era, most of them involved no graphics whatsoever. Instead, they were resolutely text-oriented, like a golf game that had you type in your swing as a number from 1 to 100.

In theory, I shouldn't have found the book very valuable. After all, I could get games written for the TRS-80s I used at home and school on floppy disks and load them in seconds, rather than laboriously entering them from a book and swatting the bugs I'd introduced in the form of typos. But I cherished it. I also typed in plenty of other programs from magazines such as *Creative Computing*, *80 Microcomputing*, *SoftSide*, and the most extravagantly programming-centric of the major monthlies, *Compute*.

The best BASIC programs published in computer magazines were surprisingly professional, in part because the bar of professionalism was easy to clear. "When I first came to *Compute* in 1983, the software that was sold, it was all handcrafted kinds of stuff in a Baggie, with a sheet or two of instructions," says Gregg Keizer, who eventually became the magazine's editor.

That made learning how to program all the more tempting: If you didn't look at much of the commercial software of the time and think to yourself "I can learn to do that," you weren't trying.

"Software was handcrafted stuff in a Baggie, with a sheet or two of instructions."

(Sure, there were people who used PCs without making any attempt to program them. But in the computer lab at my high school, in the late 1970s and early 1980s, we looked on them with pity. They were illiterate, and didn't seem to care.)

Typing in programs from listings was an intellectual exercise rather than mere rote effort, in part because you often ended up adapting them for your computer's version of Microsoft BASIC. The language had splintered into dialects as the companies that licensed it adapted it for their computers, stuffing it into whatever memory was available and improvising functions for machine-specific capabilities such as graphics and sound. It didn't look that much like Dartmouth BASIC in the first place, and it became a lingua sorta franca at best.

For instance, there's a famous one-line BASIC program for the Commodore 64 computer:

10 PRINT CHR$ (205.5 + RND (1)); : GOTO 10

That generates a random, maze-like pattern that goes on forever, or at until you press Ctrl-C. It's so hypnotic and iconic that it inspired an entire book of essays in 2012, titled, appropriately, *10 PRINT CHR$ (205.5 + RND (1)); : GOTO 10*. But it won't run on any non-Commodore computer, because its clever technique depends on the way the Commodore 64 handles graphics.

Kemeny and Kurtz were exceptionally disappointed with what others had done to their creation. In 1985 they published a book, *Back to BASIC*, which bemoaned the crudeness and inconsistency of Microsoft BASIC and other variants available for microcomputers. They titled the chapter on PC-based BASICs "What Went Wrong?" and called them "street BASICs," a moniker meant to sting.

BASIC's creators didn't just complain about what had happened to the language. They also founded a company with a meaningful name–True BASIC–which produced a version that added new features while preserving the original Dartmouth BASIC's original vision. Unlike Microsoft BASIC, True BASIC was also designed to be the same language, no matter what computer you ran it on.

"In 1983 or so, the microcomputer versions of BASIC were all different," Kurtz says. "We thought we could do some good by implementing the same language (literally) on the different computers. We were wrong. The different computers came up so thick and fast that we, a small company, couldn't keep up."

"We all know the result: books with titles like *BASIC for the XYZ Computer*. Ad nauseum."

As someone who grew up on street BASIC, I'm just as glad that I didn't know about *Back to BASIC* when it was published. Although most of Kemeny and Kurtz's beefs make perfect sense to me today, they could have used the BASIC programs I was writing at the time as case studies typifying everything that made them unhappy with what had happened to their language.

The BASIC programs I wrote for the TRS-80 would run only on a TRS-80; later, when I got an Atari 400, I wrote programs that would only run on an Atari. Like

many Microsoft BASIC programs, mine did weird, undocumented things to work around the language's limitations–most of all its sluggishness, another fact Kemeny and Kurtz weren't happy about.

My programs were not elegant; oftentimes, they weren't even intelligible to anyone except for me and the TRS-80. But they were mine, and they let me tell the computer to do exactly what I wanted it to do.

BASIC was so approachable that you could toss off little improvisational programs with barely any effort at all. I probably wrote hundreds of them in my high school's computer lab—games, utilities, practical jokes to play on my classmates. Most were meant to be disposable, and unless any of them show up on forgotten floppy disks at my parents' house, almost all of them are long gone.

But I was happy enough with one—a slot machine simulation I wrote despite never having used a slot machine—to upload it a local online bulletin-board system. Approximately 34 years later, I discovered that it was among the programs in the vast collection of Ira Goldklang, whose website is a veritable Smithsonian of all things TRS-80.

Ira reunited me with SLOT/BAS, which I loaded up on my MacBook Air, via a TRS-80 emulator. Perusing the code I'd written decades ago and then playing the game was a Proustian experience anyone who was ever obsessed with programming will understand.

Back when I was hacking BASIC, my programming idol was a man named Leo Christopherson. A junior high school math teacher who had bought a TRS-80 after one of his students brought in a BASIC programming manual, he did wonderful things with animation and sound effects in games such as Android Nim, Snake Eggs and Dancing Demon. It was all the more impressive given that he plied his trade primarily on the TRS-80, a computer with crude graphics and no official support for audio at all. (In recent years, he's recreated some of his games in modern editions for Windows PCs and Macs.)

Christopherson's techniques would have horrified Kemeny and Kurtz. His programs were rife with programming techniques deeply tied to the TRS-80's particular software and hardware, and in a quest for more speed than Microsoft BASIC was theoretically capable of delivering, he wrote more and more of his code in machine language for the computer's Z-80 microprocessor, which appeared as lines of gibberish in his BASIC programs.

"I never felt troubled by BASIC's limitations since I could use Z-80 code too," Christopherson explains. "The real limitations were imposed by the TRS-80 itself. I used to spend hours racking my brain to figure out smaller routines to do various tasks since that 16K RAM was used up pretty fast. Most anything I've wanted to do was do-able in BASIC if the computer itself was up to it."

Exiting the Spotlight

By the time Kemeny and Kurtz were unhappy enough with what had happened to BASIC to create True BASIC, the language's influence had peaked. It was no longer the default tool chosen by schools to teach programming to beginners: When I was in college in the mid-1980s, the language *du jour* was Pascal, especially among those who prized good programming practices. (I'm too embarrassed to tell you what my grade was for the Pascal course I took circa 1985; let's just say that Edsger Dijkstra would have had every right to be smug.)

BASIC had also become less of a fundamental ingredient of PC computing. It wasn't that an even easier language had taken its place. It's just that it had become increasingly possible to have a deep and meaningful relationship with a computer without ever learning to program it.

In part, that was because of the rich and powerful applications that had come along for PCs. If you just wanted to do math calculations, a spreadsheet such as Lotus 1-2-3 would do the trick even more easily than BASIC. If you wanted to wrangle vast amounts of data, a database like dBASE was built for the task.

Microsoft saw which way the software winds were blowing. "We worked early on to consolidate our dominance in languages," says Allen. "But it soon became clear that applications as tools for word processing, etc. would overshadow languages and we needed to build a word processor, database and spreadsheet. So that's how we ended up doing Word and other applications."

"When I started *PC* magazine, I declared that we would not have computer code on the pages of the magazine," says David Bunnell, who founded that publication in 1981, *PC World* in 1982 and *Macworld* in 1983. "It was ugly. I wanted it to be more of a consumer magazine."

When Apple shipped the first Macintosh in 1984, it didn't come with BASIC–an omission that seems striking to me now, although there was so much else that was interesting about the Mac that I'm not sure if anybody noticed at the time. Apple did intend to introduce a version of the language soon after the Mac's debut, but it was delayed and then eventually killed as part of an agreement with Microsoft, which released a version of Microsoft BASIC for the machine.

One moment that feels like a watershed to me happened with the publication of the May 1988 issue of *Compute*, the magazine that took type-in programs more seriously, over more time, than any other. In his "Editorial License" column, Editor Gregg Keizer announced that the magazine would no longer include such listings, a move akin to *Playboy* declaring that it would stop running photographs of naked women.

Keizer says that the decision was driven by a number of factors, including the increasing difficulty that type-in programs in a magazine had competing with slick and ambitious boxed software. "I think in the end, it was a question of how best can the magazine continue to support readers? Things were so different in 1988 than they had been in 1983 in terms of the software available and the sweat equity that people were willing to put into typing it in."

When Microsoft introduced Windows 3.0 in 1990, it became clear that the days were numbered for the text-based, command line environment in which classic BASIC flourished. But even though the company introduced a Windows version

of BASIC for professional developers called Visual BASIC–one that became extremely popular, and eventually led to a still-extant version, Visual Basic .NET–it never bundled any version of it with Windows. Instead, it continued to ship its operating systems with an MS-DOS version of BASIC called GW-Basic, later replaced by one called QBasic.

Windows Me, released in 2000, was the last Microsoft operating system to come with QBasic. By that point, it was vestigial; it only let you write MS-DOS programs, in an era when virtually nobody wanted to write MS-DOS programs.

By then, I was a former BASIC programmer myself, having started to slack off when I bought a Commodore Amiga in 1987. The only full-blown BASIC program I've written in this century was a quickie I cobbled together to automate a tiresome task in Microsoft Word. And that was seven or eight years ago.

BASIC Forever

I began this article by saying that I admire the modern-day movement to teach more people to code. I said that I miss BASIC. Which makes this question unavoidable: If everyone should learn to code, should everyone–or *anyone*–learn to do it with BASIC?

It's a question that's been asked before and has been debated endlessly. In 2006, Salon published "Why Johnny Can't Code," astrophysicist and science-fiction author David Brin's paean to BASIC's virtues as a teaching tool. He mourned its disappearance from personal computers, noting that his 14-year-old son Ben had no way to run the brief BASIC programs in his math textbooks.

"BASIC became obsolete. New innovative tools for new times."

The case against Brin's plea–made by many of the people who responded to it–involves BASIC being a quaint antique. A budding programmer learning it would be like an aspiring auto mechanic starting by learning to service a Model T.

"Eskimo has several hundred words for snow," says technologist and entrepreneur Philippe Kahn, whose first major company, Borland International, was a major purveyor of programming languages in the 1980s, including a version of BASIC. "But if you move away from Alaska it's probably not a very expressive language. In a similar way, programming saw a paradigm shift from building software for simple 'Disk Operating Systems' to designing for object-oriented platforms and BASIC became obsolete. New innovative tools for new times."

For additional guidance, I asked someone who's never far from my thoughts when I think about BASIC: Charles Forsythe. Charles happens to have been the guy I sat next to in my high school's computer lab, banging out games in BASIC circa 1980–the only one in the place whose skill I envied. Unlike me, he stuck with programming; today, he's a Java systems engineer at SAIC.

BASIC is "simple and (usually) interactive," he says. "This makes it pretty good for its original purpose: teaching beginners the basics of programming. There are other good intro languages, but with BASIC you don't have to say things like, 'We'll learn how to define a method later,' or 'we'll be learning what an object is later.'" Trying to explain a lambda expression (a.k.a closure, a.k.a function pointer) to a newbie could just get them completely confused."

Still, it doesn't sound like he's rooting for the language to reassert itself: "Once you've learned about variables and branching, BASIC stops teaching you anything that useful in today's programming world."

Brin says that "Why Johnny Can't Code" wasn't, ultimately, a plea for the return of BASIC. What he laments is that modern standard-issue computers don't offer any way at all for a beginner to turn on a standard-issue computer and immediately begin programming it, with a minimum of instruction.

"For a decade, BASIC was so universally available that textbook manufacturers put simple programming exercises in most standard math and science texts," he says. "And teachers assigned them. And thus, a far higher fraction of students gained a little experience fiddling with 12-line programs that would make a pixel move...and thus knew, in their gut, that every dot on every screen obeyed an algorithm."

In the years since Brin's essay was published, BASIC has made at least a modest comeback. He praises Nikko Ström's Quite Basic, which runs entirely in a web browser, letting you write and run BASIC programs without installing anything on your computer. As its name suggests, Lyle Kopnicky's Vintage BASIC aims to recreate the feel of the classic BASICs of his youth–with permission, he's adapted David Ahl's BASIC games to run on it.

Even Microsoft is back in the business of making BASIC useful for newbies. In 2008, it introduced Small Basic, a free, simplified version of Visual Basic designed for kids and other amateurs. It has 14 commands–the same number as Dartmouth's original version–but has wandered far from the basics of BASIC. (What you once would have accomplished with **10 PRINT "HELLO"** now requires **TextWindow.WriteLine("Hello")**.)

There are multiple BASICs available for iPhones, iPads and Android devices, too. Bottom line: If you're interested in trying out BASIC, you can, on virtually any computing device you've got. I wouldn't be surprised if that's still true another half-century from now.

None of these BASICs address Brin's original complaint, though: Unlike the great BASICs of the past, they don't stare every PC user in the face. "Remember, even one step between turning on the computer and actually typing the program will lose you 30% of the students," he says. "Add another step and you lose another 30%. Downloading and fiddling is not an option."

I agree with Brin. And I'm still glad that I learned about computers when that meant learning BASIC. However, I feel better about where we are now when I

remind myself: BASIC was never really about BASIC, or even about programming.

"The goal of Kemeny and Kurtz was to make these great, new and interesting machines available to a much broader group of minds," says Dartmouth's Rockmore. "They succeeded. Looking around at people staring at their cell phones, you could argue that they succeeded too well."

Even Kurtz seems to be at peace with the fact that few people learn BASIC these days, calling it a sign of progress: "Many of the uses of BASIC are now easily done by spreadsheets, or by specific applications. Now, practically all the functions of a modern computer can be accomplished by poking a finger at certain spots on the screen."

No, BASIC isn't a pervasive part of personal computing anymore. But the grand, improbable idea that started at Dartmouth ended up changing our culture forever–and it's tough to think of a greater legacy than that.