'Prof code'

I am always slightly baffled when I hear people referring to young people as ‘digital natives’ and greying people of my own generation as ‘digital immigrants’, because we were born into a digital world too. By the time I took my first breath in 1965, the first British business computer company (which was the world’s first) was already 14 years old and computers mediated many aspects of public life, from tax systems to weather forecasting to airliner booking timetabling to the pay and ordering systems of Lyons corner teashops. Operation over phone lines was common place and, by 1969, the first parts of the Internet were in operation. In the 1970s, many schools had computers on which interested students could learn, most being ancient cast-offs from industry. I well remember ours: a Litton, which used paper tape for storage and a magnetic drum memory in lieu of RAM. That drum had, if memory serves, 256 bytes of storage (a byte being 12 bits, in the case of this machine). Programming it efficiently meant thinking hard about the timing of the drum’s rotation, to avoid the need for the system to wait for the drum to come back round. It could not do much but I really loved programming it and spent many a lunch-break in the room containing it, with some other lads who were equally enthusiastic.

By the mid 1970s, many of us were using new-fangled microprocessors to construct and program our own computers, either from scratch or by starting with a single board computer such as the Acorn Atom, the Apple 1 or the Sym1 (below).

		A typical DIY computer of the 1970s, in this case built on a SYM1 6502 development board. I still have this machine and, despite the rats’ nest of cables between it, the KTM1 terminal (ie keyboard) and monitor driver, held together with my early teenage soldering ‘skills’, it still works.

These did not come with any software beyond, at best, operating systems allowing one to key software in: if we wanted the computers to do anything we had to program them ourselves. These early DIY machines had no facilities for high-level languages, and one programmed in machine

code. A program to print “Hi” on the screen might look like this;

A9 48 8D FE FF A9 69 8D FF FF 60

The numbers are in hexadecimal (base 16); they mean ‘load into the processor’s accumulator the value 48 (hex ACSII code for the letter ‘H’) and store it in graphics memory location 65534 {FFFE in hexadecimal) then load into the processor’s accumulator the value 69 (hex ASCII code for ‘i’) and store it in graphics memory location 65535, then return from this subroutine. I wrote the code above and the ASCII from memory, by the way, although I last programmed 6502 machine code in raw hexadecimal about 30 years ago: what you learn as a young teenager you tend not to forget. The Lytton was similar, though it was programmed in octal (base 8) numbers that were represented as holes on paper tape.

As time went on and memory became cheaper (so we could save our pocket money and aspire to, say, 4 kilobytes!), we improved our systems, adding higher level languages like Forth or BASIC and sharing our code. Some of us also constructed modems for operation over the phone lines, allowing us to connect to distant servers and also to connect to the Internet (I had better not specify how free internet connection was achieved, for reasons that will be obvious to any readers who were doing the same. Back then it was not exactly illegal, but in a sort of grey area and people turned a blind eye as we did no damage - ‘hacking’ did not in those days mean being a vandal but just meant improvising and repurposing technology in a clever way). Simple services like Telnet allowed us to read, for example, the news pages at NASA, and Gopher allowed internet searching (this was all long before the World Wide Web). By the mid 80s, social media were available and much-used through the medium of bulletin boards: I still have my e-mail address at the WELL, the hippyish Whole Earth ‘Lectronic Link begun in 1985. There, one could learn quite a lot about the world through the ‘alt.’ boards (again, older readers will know what I mean, and will remember just how mind-broadening was the writing by the very free, liberal-intellectual contributors to these boards and will also remember the respectful, ‘troll’-free discussions).

One thing that divides those of us who grew up interacting with the digital world of the 20th century from those who grew up in the21st is that we could all write code (we had to – we had no choice: most of us were pretty handy with a soldering iron for the same reason) whereas a shockingly high proportion of even science graduates nowadays cannot. The students who *can* code mostly learn

formally, rather than from club newsletters in teenage bedrooms, and they learn the latest techniques as their starting point. Thinking in an abstract way about objects and functions is second nature to them (or, rather, it is first nature) and they have no reason at all to have any mental pictures of the latches and shift registers deep in the heart of the central processor, nor (for most purposes) of orders and timing of events. And this makes for a second division between the young and older generations, at least the older amateurs in computing who, like me, have not really kept up. I still think of code procedurally, as a sequence of instructions to be followed in a specific order, and I have a mild aversion to using 'objects', finding it more natural to handle every operation explicitly. The code I write for simulating biological systems to aid my lab work does work, but properly trained scientific coders tend to look at it with wide-eyed amazement. Comparisons to flint axes and cave paintings are not unknown.

Recently, I was invited to give an evening talk about the lab's synthetic biology work at Imperial College and, in the wine-and-nibbles session afterwards, I was talking about writing programs with Dick Kintney, a world-leading biological engineer who is native to that parish. I mentioned the way my code amuses young programmers, and he recognized the phenomenon at once, being of a similar generation and also being someone who uses programming as a tool rather than an end in itself. Laughing, he told me that his own graduates have a special name for it, a name I will now borrow.

From now on, when someone asks what language I use for writing simulations, I will be honest: I will tell them I write in 'Prof code'.

Jamie Davies
Edinburgh
May 2017

Links:
LEO, the first business computer:
https://www2.warwick.ac.uk/services/library/mrc/explorefurther/digital/leo/story/

Dick Kintney's web page:
http://www.imperial.ac.uk/people/r.kitney