Alex Groce (agroce@gmail.com), Northern Arizona University

Brian Harvey's *Computer Science Logo Style (Volume 1: Symbolic Computing, Volume 2: Advanced Techniques, Volume 3: Beyond Programming)* begins with the words: "This book isn't for everyone." There follows a brief account of the fact that not everyone needs to program computers, based on an economic (Marxist-flavored) tirade (that I mostly agree with). The closing of the introductory paragraphs is the part that matters, though: "This book is for people who are interested in computer programming because it's fun."

Sometimes someone will ask me what programming language is best for introducing a kid to "coding" and I'll mumble "I dunno, Scratch is what I hear people are into now, it's probably good, people I respect seem to be doing good work on it…" or something, but my heart isn't in this answer, which is false answer, easy answer, safe answer. I know the true answer. The true answer is "Drop all this nonsense, and teach them Logo, but without over-emphasizing just the silly turtle."

That's how I got my start, though the original source was not *Computer Science Logo Style*, it was various Apple/LCSI Logo books (there were, if I recall, and the Internet does not seem to be able to help my clarify my memories, two books with ring binders, one a reference manual, and one a kind of "get going!" guide), that offered lots of examples based on string (word, in Logo lingo) and list manipulation. That "GANDALF backwards is "FLADNAG will probably be the last thing I remember when senescence takes my mind away. But never mind me; *Computer Science Logo Style* wasn't published until 1997, so it came too late to help me, but I was raised on what I think of as precursors to its joys.

Logo is, basically, LISP, but, as the Coen brothers might put it, "you know, for kids." The syntax is more BASIC, less s-expression purity, but the ideas of providing features for interactive development, a large variety of list processing primitives, the ability to easily run generated code, and some aspects of the functional style are all there--but with the target audience being schoolchildren, rather than Richard P. Gabriel and company. The language's history (https://dl.acm.org/doi/10.1145/3386329) is interesting and complex, but the key point for this review is that, for many years, Brian Harvey was a Logo implementer and evangelist. His UCB Logo is still available, with an active GitHub repository maintained by Harvey's successors at https://github.com/jrincayc/ucblogo-code. From this experience, he produced *Computer Science Logo Style*, which, using Logo, presents an idiosyncratic walk through a large chunk of computer science.

Computer Science Logo Style is, simply, fun. It is, as noted, idiosyncratic; some of Harvey's metaphors are not ones I would have chosen, or ones I can fully agree with. His approach to automata theory, or programming language design, is not mine. But these are quibbles. The three volumes offer teenagers, precocious children (for example, any nine year old who has made it through a Camiroi primary school up to that point could easily handle all three books), and non-"computer person" adults a way to see much of the beauty, excitement, and cleverness

of computer science, without fakery, education-theory-of-the-moment, or the slightly acrid odor of “it’s for your own good, so you can get a job coding.” Harvey doesn’t really care what job you get; Harvey wants to show you something fun, the way he’d have you listen to his beloved Beatles, or watch Animaniacs (in his view, the only good thing on TV, perhaps ever). This is not to say that going through the books won’t give you a serious education, or even broaden your understanding of various things if you are already fairly informed about computer science. When I came across these books in the NC State library, I was a fourth-year computer science major who’d written a Pascal compiler in C; but I learned a lot of things, and unlearned a few others. The first volume already covers higher-order functions (map, reduce, and filter!), advanced tracing and debugging, top-down vs. bottom-up program design, and depth-first and breadth-first search approaches, including search tree pruning. And that’s just the first volume. By the middle of volume 2 you’ll be implementing a data-driven (via property lists) version of ELIZA:

```
to analyze :sentence :keywords
local [rules keyword]
if emptyp :keywords [norules stop]
make "keyword first :keywords
make "rules gprop :keyword "rules
if wordp first :rules ~
  [make "keyword first :rules make "rules gprop :keyword "rules]
checkrules :keyword :rules
end
```

Now, *Passages* is in theory about books that are particularly interesting to software engineers, and I think it’s time to justify CSLS (as I shall henceforth refer to *Computer Science Logo Style)* for that audience. The books chosen for this column are often either obviously of interest to everyone under the sun (sure, people who don’t care about computers can enjoy and learn from Confucius and John McPhee), or pretty clearly really for computer science people, but conceivably readable by a dedicated layperson (Brooks, Agans, Hunt & Thomas). CSLS is unusual in that it is very clearly ideal for the educated non-computer-scientist (aged 10-120) who wants an introduction to the field more suitable for becoming an amateur programmer and enthusiast (we had them, back in the 1980s, I have heard) than a would-be professional with a GitHub portfolio angling for a “good jerb” offer. Is CSLS worth reading for a seasoned software engineer?

I would argue yes. The reason is Harvey’s many, many (and, pleasingly, often non-mathematical: this is not a book where you implement Newton’s method 20 ways) extended programming examples, and his attitude to program design, including, essentially top-down vs. bottom-up methods. Chapter 13 of the first volume explicitly dives into the concept, using the campuses of Stanford and Berkeley (where Harvey taught for many, many years) to exemplify the two approaches. However, the value and insight for the seasoned program designer is more in the sum total of all the programs and explorations over the three

books than in the explicit discussion. I think that an appreciation of Harvey's Logo programs can inspire an examination of one's own code and design instincts, in terms less loaded with the jargon and ephemera of the moment than reading any contemporary design manifesto. That's worth the time spent reading three very fun volumes of writing about computer programming.

The CSLS books are freely available online (https://people.eecs.berkeley.edu/~bh/v1-toc2.html), and the first volume is readily available in a handsome paperback from the MIT press. The other two volumes seem to be out of print, but are not hard to find.