Alex Groce (agroce@gmail.com), Oregon State University

Daniel P. Friedman and Matthias Felleisen's *The Little Schemer - 4th edition* has a goal, and states that goal up front. The book exists in order "*to teach the reader to think recursively*." From the point of view of this column, that raises two questions. One, doesn't the software engineer (you) already know how to think recursively? Two, should we care if the software engineer knows how to think recursively? That the book raises questions is not a problem. *The Little Schemer* itself, other than the introductory material and index (and charming illustrations by Duane Bibby) is in the form of a Q&A, with answers given. That is:

Is it true that *The Little Schemer* is entirely in Q&A form? It is true (with the exceptions noted above).

Is it true that *The Little Schemer* uses examples that are mostly about food? It is true. I will quote the preface. "Food appears in many of our examples for two reasons. First, food is easier to visualize than abstract symbols. (This is not a good book to read while dieting.)... Second, we want to provide you with a little distraction… a little distraction will help you keep your sanity."

Is it true that a Q&A about the basics of a programming language not widely used in production systems, using food-based examples, is not obvious reading for a software engineer? It is true.

Is it true that the software engineer knows how to think recursively? It is true that the software engineer has usually encountered recursion in a class, if they are formally trained in computer science. Even if not, the software engineer, whoever that is, probably knows the basic idea, or at least has seen recursive Tower of Hanoi or quick sort code.

So, is it true that the software engineer knows how to think recursively? It is sometimes true; it depends on the engineer, the class (if that is how the engineer learned to think recursively), and the language. There is a difference between knowing about an idea and knowing how to think with it.

Why do you keep answering things with "it is true" or asking "is it true?" It sounds weird. It is true that is sounds weird; I do it because it is something they do in the book, because many of their Q&A pairs are basically also the output of a Scheme interpreter, and the question is going to yield Boolean true as the result.

Do many Java and C/C++ based classes essentially treat recursion as a cute gimmick that is not much use in the real world, but you better know about? It is true, alas.

Is it true that software engineers should know how to think recursively? I think so. Even if you write Java code, or embedded C, and never implement anything via recursion, recursive thinking is still the most effective way to define just about anything in computer science. A large chunk of software engineering work (static analysis, testing, etc.) is based on essentially recursive ideas. So the answer is: it is true if you think software engineers should know some actual computer science.

Is a software engineer a computer scientist? Software engineering uses computer science, which includes programming language theory (which includes, in my opinion, learning a LISP, which is what the book we almost forgot we were talking about does), the way a civil engineer uses the scientific knowledge of statics and dynamics. So no, but also yes.

Can you convince me more effectively? I hope. I would claim that a lot of the progress in software engineering productivity (a problem David Parnas considers suitable for a small team to tackle for a week in 1972 can now be effectively handled by a lone talented experienced individual in less than half an hour) is due to advances in programming languages. One of the basic paradigms of programming is functional programming, and LISP is the *ur*-text there. Scheme is just LISP in a snazzy new suit (the first version of *The Little Schemer* was *The Little LISPer*). Now that every language and its grandmother is getting a lambda mechanism, you ought to even be able to claim that this is all very relevant to remaining a fashionable *au courant* on-the-mark skill-growing bootstrapping self-motivated software engineer. You could probably put it on your resume, in suitable disguise. I would personally rather do it because it is fun, but not everyone shares this reviewer's motivations.

Is it true that this book is rather basic? It is true. If you already know Scheme or LISP, you will not encounter a huge amount of new material here. The coverage of the really important PL concepts, like higher order functions, is not deep. This could introduce someone to programming (and would not be a terrible way to do so, depending on the someone).

Is it readable for more experienced programmers? Yes! The Q&A format and novelty of the content, including the focus not on tedious numeric examples, but on food, glorious food, makes the book a quick and interesting read for even experienced software engineers. It is more fun to manipulate bacon, lettuce, and tomato (or, rather, (bacon lettuce tomato)) than to implement Newton's method. At least it is to me.

If I like this book is there more to read? Yes, you can continue to *The Seasoned Schemer*, *The Reasoned Schemer*, and *The Little Prover*. *The Seasoned Schemer* is by the same authors, and is a true sequel, carrying on where this book leaves off. It is equally fun, and covers things like higher-order functions in a much more thorough (if still strange and not at all like a typical textbook) way. *The Reasoned Schemer* is just over 10 years old, has a different second author, and covers logic programming (they call it relational programming, which may be a better name) in the same engaging, food-based style. *The Little Prover* is basically brand new, and covers automated theorem proving, which is usually the topic of advanced graduate classes and very dry books published by Springer, but here is, again, in the form of a Q&A about flapjacks.

What about the general reader? A long time ago, in your first column, you suggested books the general, educated reader might enjoy as a filter for *Passages* classics. I am dubious about some of your choices (is anyone who doesn't program going to really like *The Pragmatic Programmer?*). Is this another dubious choice? It depends. If the general educated reader does not like bacon, lettuce, tomato, or peanut butter they might dislike this book, but it would show a lack of taste. I think it's a great book to give the general reader you want to convince computer science is more than a grim and unpleasant business.