Book Reviews

MARK LAMOURINE

Fluent Python
Luciano Ramalho
O'Reilly Media, 2015, 474 pages
ISBN 978-1-491-9-46008

For a long time, I've told people I only need one book on Python: David Beazley's *Python Essential Reference*. I don't need tutorials any more—none of the references I've read does a better job than Beazley of balancing completeness and compactness. In *Fluent Python*, Ramalho has given me both a second book to keep close and an archetype for a type of book that I would like to see more of.

Ramalho sets out to teach not just what Python can do but how it works. Python has a lot of history, and the feature set is a mixed bag. It inherits from a lot of sources, and Ramalho is familiar with them and talks about them where it adds to the reader's understanding.

The thing I really like about *Fluent Python* is that Ramalho talks about the characteristics that make Python special, different, and, especially, expressive and readable. He devotes a lot of space to practical aspects and to idiom.

This isn't a book for beginning programmers or even developers just starting to learn Python. While there are echoes of the topics you'd expect to see in a programming language book, they are treated from the standpoint of internals and language-design choices. In many cases, Ramalho addresses would-be language purists on their own terms explaining why the language developers made their choices and how those affect Python operation and performance. One standout is how Python has adopted functional programming concepts, and how the traditional constructs (filter, map, reduce), while they exist, are largely better written using list comprehensions in Python. Ramalho explains how comprehensions in Python are both more clearly expressive and more efficiently implemented than a classic MapReduce construct.

The author doesn't shy away from what even he considers to be warts on the language. Python's syntax restricts the lambda construct in ways that make anonymous functions nearly useless. He shows instead how to use regular functions, which, while more verbose, are often clearer for the reader and developer trying to debug a set of deeply nested anonymous functions. Developers coming from other scripting languages also face difficulties that arise when trying to extend built-in types. In this, I agree with him that the seeming problem is, in the grand scheme, a good thing, discouraging people from trying to do things which lead to obscure or too-clever code.

I especially liked the sections on decorators and his coverage of iterators and generators. I've often seen tutorials on the syntax for creating and using both of these constructs, but Ramalho discusses both the theory behind them (decorators are closures? Oh!) and how they behave in operation. I find the under-the-hood aspects to be useful when I'm deciding when to use constructs like these.

At the end of each chapter, the author includes an extensive references section and, my favorite, a "Soap-Box" section where he talks about his preferences, biases, and impressions. These give the reader both a sense of his background and some input on topics that can be interpreted as opinion (or religion).

These days I mostly skim books and then set them aside. *Fluent Python* is one I mean to revisit. It's too meaty to completely digest in one pass. Now I know where to look when I want to learn more about Python's more interesting possibilities.

Once Upon an Algorithm
Martin Erwig
Massachusetts Institute of Technology, 2017, 317 pages
ISBN 978-0-262-0-36603

I was both intrigued and dubious when I first picked up Erwig's book. I like the idea of using metaphor and, especially, storytelling to make technical subjects accessible. I like to use them even when talking to my peers since a good metaphor can often be a shortcut to understanding. Presenting technical topics this way, however, risks oversimplifying. Such a presentation can either give a clear but incomplete treatment or bend metaphors so badly in an attempt to be make them rigorous that they lose their relevance. You can only push stories so far when applying them to complex topics.

Erwig starts off simply enough: Hansel and Gretel mark their path and then find their way home. They do it by following a series of repeated steps, first marking their path with stones and later bread and then following the markers back. This is the kind of thing I'd expect in a popular treatment of algorithms. The vocabulary and writing style is at odds with the simple story. Erwig is using children's stories, but he's not telling one.

It turns out that *Once Upon an Algorithm* is aimed at neither the purely technical nor the broad popular audience. Instead the intended audience, one that I am part of, is outside the field of computer science: the curious, dedicated lay reader.

The author organizes the book into two sections: "Algorithms" and "Languages." This works because, contrary to my expectations, he's not trying to explain algorithms by coding them. He actually treats the Theory of Computation right from the beginning, using the stories and the algorithms they illustrate to introduce the deepest concepts of computation: abstraction, representation, complexity, and computability.

As noted, in the first chapter, Erwig uses Hansel and Gretel to conceptualize an algorithm as a set of steps that can be followed to achieve some goal. In the second chapter, Sherlock Holmes is used to illuminate modeling, data representation, and abstraction. In the third, Indiana Jones is the focus used to discuss searching and sorting. In all three chapters, Erwig ends by talking about the deep questions that arise when trying to represent the real world with mathematics, logic, and machines. The third chapter closes with the best non-mathematical description I've read about the meaning of *P* (the set of problems computable in polynomial time), *NP* (problems where a given solution can be tested in polynomial time), and why the idea that $P = NP$ (or not) is important for computation.

As if that's not enough, the second half of the book covers the theory of formal languages. Erwig uses the song "Somewhere Over the Rainbow" and the example of musical notation to show how ubiquitous "computation" is. In a very real sense, a musical score is a "program" that can be converted into a result (a performance) by a computer (the conductor and musicians). The movie *Groundhog Day* serves to show how iteration and looping work (and why terminating conditions are so important). *Back to the Future* is the inspiration for a discussion of recursion, and *Harry Potter* serves as the backdrop for the final chapters on the theory of abstraction and types.

In the end, I found the discussion to be on-point and clear. Erwig doesn't condescend to the reader despite how easy that would be given the thesis that common stories can illustrate the theory of computation. He shows in this way how computation isn't really some strange esoteric field but is grounded in everyday ideas and activities that anyone can relate to. The title of the book might lead someone to expect a watered down popular "dummies" treatment, but that would be a mistake. Erwig does indeed know his audience and writes to them. That audience will be well served by *Once Upon an Algorithm*.