Book Reviews

ELIZABETH ZWICKY, WITH MARK LAMOURINE AND TREY DARLEY

Regular Expressions Cookbook, Second Edition

Jan Goyvaerts and Steven Levithan

O'Reilly and Associates, 2012. 575 pp.

ISBN 978-1-449-31943-4

This is an excellent reference work, which will some day—perhaps many days—save you untold effort. Yes, this book goes over how regular expressions work, but where it shines is in providing practical recipes that take into account not only the details of regular expressions but also the details of the world. For instance, in processing ZIP codes it notes that there is one ZIP+4 (and only one) that contains letters, but then notes that your mail to Saks' shoe department will deliver just fine without it anyway, and recommends you just ignore it. *Regular Expressions Cookbook* is happy to suggest combining regular expressions and code for readability and performance.

The book is admirably agnostic, bearing in mind the possibility that you will want to deal with phone numbers and postal codes from outside the US, use non-ASCII character sets, and parse Windows-specific values. Although it is impossible to cover all the languages and situations where you may want to use regular expressions, it covers a good wide variety, including uses in text editors, and provides references to useful testing tools. I might not have picked up this title had I not been looking at books to review (after all, I already own two books on regular expressions), and that would have been a real loss. Even if you're already a pro with regular expressions, this book will point out details and save thought; if you're not, it will help you without making you too terribly dangerous.

Python for Data Analysis

Wes McKinney

O'Reilly and Associates, 2012. 432 pp.

ISBN 978-1-449-31979-3

This is a specialist's book. If you read the title and think, "Wow, how handy; I have this data I know how to analyze, and I know some Python, and learning all of R seems a bit unwieldy when I could do all my processing in Python," then you really want this book. If you are fully confident in your skills in one thing or another, either Python or data analysis, and you're interested in teaching yourself the other with a bit of assistance from a reference work, this title would still be a good choice.

If you need hand-holding, move on. This is the kind of the book that says airily that there are many ways to get a random sample of items, with different performance implications, and then provides an example of exactly one of them. You are expected to already know what performance implications it has and to think of the rest for yourself. (It's hardly an unusual problem, after all.) The book also, in the Macintosh installation instructions, tells you to download a software package without specifying where you would download it from. (For one thing, the answer is easily findable in search engines, and for another, it already told you in the Windows instructions—why you would read the Windows installation instructions in order to do a Macintosh installation, I do not know.)

I'm probably going to use my copy, if I can pry it out of the hands of the Python guy at work who has been asking me wistfully for months whether I know anything about pandas (the Python library, not the bamboo-eating animals).

Managing the Unmanageable: Rules, Tools, and Insights for Managing Software People and Teams

Mickey W. Mantle and Ron Lichty

Addison Wesley, 2012. 406 pp.

ISBN 978-0-321-82203-1

There are some good insights in this book and some pithy rules of thumb; it's an approachable book about managing programmers, which will probably help many managers, especially those who manage groups composed entirely of programmers turning out new projects. All the same, I couldn't love it. Some of the problem was the authors' style,

which doesn't work for me (and that's a highly personal thing, so you should check out the book to see how you feel about it yourself). Some of that was the laser-like focus on traditional programming. The authors are quite condescending about people who program in scripting languages or, worse yet, use GUI tools, and they don't care about non-programmers—including QA, system administrators, designers, and technical writers—at all. Apparently in their world, programming managers don't deal with such people. Also programming managers only manage development groups, not support groups.

The book does a much better job than most on the nitty-gritty of interviewing and hiring programmers, and the rules of thumb it presents get a nice wide range of perspectives represented. If the style and the tight focus work for you, this book is a good place to start in the programming management game; the content strikes me as mostly right, if occasionally over-opinionated.

Python for Kids

Jason R. Briggs

No Starch Press, 2012. 313 pp.

ISBN 978-1-59327-407-8

Super Scratch Programming Adventure!

The LEAD Project

No Starch Press, 2011. 158 pp.

ISBN 978-1-59327-409-2

These two books take superficially similar approaches; both of them use video game development to motivate kids to learn to program. *Python for Kids* is aimed at kids age 10 and up, whereas *Super Scratch* is geared toward kids 8 and older; however, even apart from the age difference, these books will suit radically different children.

Super Scratch is in a comic book format, and it focuses on a language designed for children. *Python for Kids* is a standard introduction to Python, gently modified for children. For a lot of kids, particularly kids on the younger end of the age range, *Super Scratch* is going to be the more attractive option. *Super Scratch* starts off with an intergalactic adventure, and gets to making a cat move on page 21 (and that includes 10 pages kids can skip and a couple in which you're already looking at the cat on your screen). *Python for Kids* gets halfway through before it starts covering a game, and it begins by adding numbers. If your kid wants to program for programming's sake and is likely to be offended by having things dressed up with irrelevant space-going comic strips, *Python for Kids* is the better choice.

Of the two, I think *Super Scratch* does a better job of bringing programming to kids because it talks about debugging, for example, and does a better job of providing questions kids are likely to be interested in answering. Of course, *Super Scratch* also starts with a programming language designed for kids, which is a major leg up.

Python for Kids has to work within the limitations of Python, which requires a certain amount of typing and discussing integers and the like. On the whole, I think *Python for Kids* copes pretty well, although my head exploded at the paragraph "Why is a giraffe like a sidewalk? Because both a giraffe and a sidewalk are things, known in the English language as nouns, and in Python as objects." OK, first of all, a giraffe is not a noun. The word "giraffe" is a noun. Second, there is no guarantee that nouns are things or things are describable with nouns. "Beauty" is a noun, but beauty is not a thing, and a pregnant giraffe is a thing, but only describable with a noun phrase. Third, objects, nouns, and things have very different characteristics. A giraffe is even less like a Python object than it is like a noun. Fourth, while giraffes and sidewalks are like each other in their degree of dissimilarity from both nouns and Python objects, this totally fails to illuminate me about Python objects and doesn't come up again. Presumably, if I were 10 years old this would bother me less, but I still don't think it would do much to help me understand Python objects.

My test child is 8; she has encountered *Python for Kids* in its previous online existence, and by all reports was unimpressed. (Like me, she is not interested in programming for programming's sake, so she's pretty much out of its target audience in several directions.) She was quite taken with both *Super Scratch* and the Scratch programming language, and although she required a little help to make the connection between the book and the screen, she was enthused about working with it. At which point, using only the instructions she could not proceed without in *Super Scratch*, she carefully recreated in Scratch...the first turtle drawing exercise in *Python for Kids*, which she ran into at least six months ago. Go figure.

Meanwhile, these experiences seem to have communicated only some of what they were trying to. Days later, we looked at the screen saver on my computer, drawing fancy flowers, and I said to her, "You know that's a computer program, right? People write programs that draw flowers." "Really?" she said. "Huh. I've written three programs, you know." Score a point for empowerment; take it away for not having connected that experience to the things computers do that she loves.

—Elizabeth Zwicky

Assembly Language Programming: ARM Cortex M3

Vincent Mahout

Wiley-ISTE, 2012. 246 pp.

ISBN 978-1-84821-329-6

I'm one of those people who thinks that software developers should be aware of the workings at least one and probably two levels below where they are working. That would be reason enough to want to read up on assembly language. The recent growth in consumer and hobbyist ARM systems makes that a good selection.

Modern compiled and scripted languages plaster over so much of the arcana that goes on at the machine level that there's no good place to just jump in and get coding. Mahout takes about five chapters to get to some working code. Those chapters cover the ARM architecture and elements of assembly syntax.

The final four chapters are where this book earns its keep. Chapter 6 demonstrates how to implement logical constructs such as looping and branching blocks that in a high-level language might be represented with a single keyword and a couple of curly braces. Chapter 7 covers modularity and constructing procedures and functions, including detailing the ARM-calling convention. Chapter 8 is about handling hardware- and software-generated exceptions. Chapter 9 walks through the creation of a complete simple program, detailing each of the steps required to assemble, link, load, and run the program. Remember, in assembly you're responsible for initializing the stack and all of the memory you've allocated before branching to your program.

Aside from the long exposition that must happen before getting to the meat, this book has several other quirks that effect the reading experience. The contrast of the graphics and code typesetting detract somewhat from the otherwise clean layout. The code boxes use an unnecessarily dark background that makes the black text hard on the eyes. Many of the graphics appear to be color images converted to gray-scale without any additional touch up.

There is, throughout the book, an odd use of language, at least to my American English ear. When describing the sample project used to illustrate the use of the assembler/linker/loader tool chain, Mahout begins, "This entire project is of restricted algorithmic interest." I probably would have chosen "limited." The word choice doesn't confuse the meaning but can stand out as you read. If this issue had happened once I would have passed it off as a quirk, but it occurs repeatedly. Mahout is a native French speaker. The book is published and printed in the UK. I would have thought that an English-speaking editor would have spent a bit more time polishing simple word choices.

The number of ARM family variations and the fact that ARM SOC (System on a Chip) are manufacturer-specific mean that Mahout can't talk about things outside the core spec itself. He chose a fairly recent mobile core, the Cortext M3, as his working model.

In the same way that there are different flavors of compiler for high-level languages, there are multiple assembler environments for a given processor family. Mahout based his book on the Keil ARM-MDK (Microcontroller Development Kit). Kiel has been purchased by ARM, and the "Lite" version is available from the arm.com Web site for free and is capable of demonstrating all of the work in the book. Appendix D of the book details how the GNU-GCC assembler (specifically the assembler from the Sourcery G++ suite) differs from the ARM-MDK.

This is certainly not a book for a novice programmer. If you need proper ARM references, the ARM site itself has those for each of the processor flavors, and for a specific SOC you will need the manufacturer references. I don't want to recommend against this book for an experienced coder who wants to taste assembly language or get a look under the hood of an ARM system, but I will warn that reading it will take some dedication. This might be a good book for the classroom, but I would hope that the teacher would re-organize or gloss the early chapters and somehow get the students straight into some hands-on work. I'm still looking for the K&R or Stevens of modern assembly.

Super Scratch Programming Adventure!

The LEAD Project

No Starch Press, 2011. 158 pp.

ISBN 978-1-59327-409-2

Since the invention of Logo and the turtle in 1967, people have been trying to create languages and environments that invite kids to learn and explore programming. The Scratch programming environment was created at the MIT Media Lab's Lifelong Kindergarten project in 2006. An environment like Scratch still has to be presented to kids in a way which helps them engage.

Super Scratch Programming Adventure is published in North America by No Starch Press, but was developed and written by The Lead Project, a collaboration between the Hong Kong Federation of Youth Groups and the MIT Media Lab.

When I got this book in the mail, the first thing I did was set up Scratch on my 13-year-old daughter's computer. After supper I handed her the book and walked away. My daughters have both been resistant to learning programming from me and I generally don't push except occasionally to offer some

new toy to try, like this. Several hours later she was still playing with Scratch. I'll call that a win. She continued to play with it on and off for several days.

When I asked her what she thought of the book she said she liked it in general. She thought the comic book presentation was a bit young for her, but that it didn't detract once she got into it. She played with each of the games and explored some of the variations, but she didn't follow the progression of the book faithfully and she didn't formally complete any of the "lessons" in the way the authors intended. She said that a big part of Scratch is creating the artwork for the stories. She doesn't consider herself an artist so she stopped when she ran out of things to do with the (large) provided set of "avatars."

With the experiment over I started working through the book myself.

Scratch is a programmable storytelling environment. The user can draw characters (avatars) and backgrounds or use some from the provided library. The stories are programmed by dragging and dropping a set of tool bar objects, representing logic constructs and methods, on various other objects, such as avatars or drawing pens. A loop or code block actually wraps around the contained steps so the nesting and scope are visually clear. Method parameters are text boxes whose contents the user can change. Types of programming objects (logic, avatars, drawing tools) are color coded. Scratch and the programming examples for the book are available online from the URLs provided.

Super Scratch Programming Adventure! has the typical cast of characters: the human, for the reader to identify with, and a collection of animals and aliens to play the roles of helpers and villains. The adventure is presented as a series of crises to be overcome. Each crisis has a program that starts out working, but not in the desired way. The text guides the reader through the process of changing the program to solve the problem. The end of each chapter suggests some other ways to experiment to see the effect of different changes.

The chapters present the typical concepts of variables, code blocks, looping, and procedures in a purely practical and experimental way without any attempt at theory. The students get a visceral understanding through their play. In a classroom setting a teacher might have a discussion session to get the students to talk about the implications of what they've done, but that's not part of the text. By the end the students have played with 2D motion, sound, color, and user interaction.

As I mentioned, Scratch is a storytelling environment and *Super Scratch Programming Adventure !* is a storybook. Storytelling isn't much fun without an audience. Scratch provides a means to upload stories to a public Web site, and the book encourages the student both to do that and to explore the stories there for additional ideas.

My experiences with recent middle and high school "computer" classes have been disappointing, and I expect it's not uncommon. Recent activities in the UK [1, 2] and this book from Hong Kong (not to neglect any US efforts I'm not aware of) give me hope that middle and secondary computer education may yet grow beyond teaching proprietary word processing software. This book is probably best suited to a middle school environment. It's going to require creative and enthusiastic teachers to foster the sense of expressive freedom needed so that the students never know they're "programming." I'd certainly recommend this book and Scratch to an involved parent whose child has expressed an interest in using computers for something more than viewing videos and playing games. This book will stay on my daughter's shelf, and it may yet call her back to play.

[1] http://www.guardian.co.uk/politics/2012/jan/11/michael-gove-boring-it-lessons.

[2] http://www.raspberrypi.org/about.

—Mark Lamourine

The CERT Guide to Insider Threats: How to Prevent, Detect, and Respond to Information Technology Crimes (Theft, Sabotage, Fraud)

Dawn Cappelli, Andrew Moore, and Randall Trzeciak
Addison-Wesley Professional, 2012. 432 pp.

ISBN: 978-0-321-81257-5

Carnegie Mellon's CERT Insider Threat Center has (in collaboration with various law enforcement agencies) amassed a substantial data set of criminal cases involving malicious trusted insiders. Through analysis of this database the authors (all of whom work for the Insider Threat Center, by the way) have identified distinct profiles associated with fraud, IP theft, and sabotage. The authors use these case histories to great effect throughout the book to drive their points home.

They won my heart early with this line in the book's overview: "If you learn only one thing from this book, let it be this: Insider threats cannot be prevented and detected with technology alone." For managers, faced with a difficult and a subtle problem, the temptation to throw an expensive black box at it, put a tick in the box, and assume that it does what it says on the tin can be irresistible. Couple that with the trend of outsourcing critical functions and you've got a recipe for danger.

The first four chapters provide a fairly high-level overview of case histories, profiles, motivations, and mitigation strategies. The rest of the book is devoted to issues specific to the software development life cycle, best practices for prevention and detection, suggested technical controls, and in-depth examination of selected cases. Technical types can glean useful insights from this book, but to get the maximum benefit, try organizing a reading group with the folks over in HR.

Advanced Internet Protocols, Services, and Applications

Eiji Oki, Roberto Rojas-Cessa, Mallikarjun Tatipamula, and Christian Vogt

Wiley, 2012. 260 pp.

ISBN: 978-0-470-49903-0

I marvel that such a slender volume can pack such a wallop of disappointment. Based on the publisher's description, this book sounded like it would pair nicely with the new edition of *TCP/IP Illustrated, Volume 1*. I hoped it would fill in the gap on topics that Kevin Fall omitted for brevity's sake (i.e., dynamic routing protocols, traffic shaping, QoS, and so forth). Sadly, this book contains so many errors (both linguistic and technical) that I cannot imagine an editor was ever even in the same room with the manuscript. This is a rambling 260-page paraphrasing of RFCs that somehow manages to be less readable than the RFCs themselves. This book lists for $US 99.95. For that amount of money you can buy two copies of Fall's opus. Do yourself a favor and skip this one. Hopefully, Fall is hard at work updating *TCP/IP Illustrated, Volume 2*.

—Trey Darley