Learning Python

Quantum books interviewed Mark Lutz, author of "Learning Python ", whose 3rd edition has recently appeared. For this edition of our newsletter, Mark – a world-class expert in Python - talked with us about this book and about his experience with the Python language itself.

1) How did you become interested in Python, to the extent of writing several books about it over the span of more than a decade?

My Python history goes back 15 years. I first found Python in 1992, when I was still a frustrated C++ programmer. At the time, I was being paid to explore available scripting languages for use in a GUI builder system; Python turned out to be the best tool for the project. In fact, Python was so much better than the alternatives that it completely took over my career, and wound up consuming much of the next 15 years of my life.

Before finding Python, most of my background was in large-scale software development (compilers and the like), but I also spent some time in applications development. I suppose I saw enough bad code and bad decisions in this field early on, that finding Python was a sort of breath of fresh air. Its emphasis on simplicity, quality, and productivity was in stark contrast to the norm. To me, Python seemed to shout that we could do better in the software field. Given my background (along with a nasty case of idealism), I'm inclined to believe that any quality improvement in this field is worth the effort. I'd even say it's a responsibility, if we agree that software is engineering.

Because I got involved with Python in its "early days," I was able to sign up to write the first Python book in 1995, and start teaching the first Python training classes in 1997. The first Python book, Programming Python, actually took a lot of lobbying. At the time, O'Reilly was hesitant to take a chance on Python because it was so new and obscure, and in fact turned it down at first. They were worried it might appeal only to hardcore geeks. (This was at a time when downloading Python often meant dealing with a collection of uuencoded email messages; remember that?) They agreed to sign the book only after about 6 months of being pestered by me. Of course, it's grown into a huge market since then, largely because they took the chance.

Training was something I more or less made up as I went along; basically, someone called me from Fermi Lab in Chicago looking for a class, and I had some vacation time to burn at my day job. In the decade since then, I've taught over 200 classes around the world, and have been a self-employed trainer for the last 9 years. Not bad, I suppose, for something that I never really planned.

Training and writing are both intense work, and people often ask me what motivated me to do it. I suppose part of it was the excitement of Python's early days. Helping to promote and popularize Python in the software field was very much like being part of a start-up company, and the Python community never seems to have lost that spirit entirely. It's gotten larger and a bit more formal, of course, but it's still an exciting project to be a part of today.

Really, though, my primary motivation is probably just that Python is so much fun to use. Apart from book and teaching examples, I haven't gotten paid to write software for almost a decade, but I still often find myself picking up Python for the pure pleasure of programming. Fun matters too, and Python has that in spades over the scores of other languages I've used in my career. Honestly, I'd rather flip burgers than go back to using something like C++ again; to me, it's just too tedious.

2) How would you describe Python's niche in the ecosystem of existing programming languages? What would you say are Python's particular strengths, when compared to other languages?

Well, after teaching Python to over 3,000 students and hundreds of companies and organizations, I'm not sure I'd say that Python really has a single niche in the industry. It's used everywhere – in web site development, hardware testing, numeric analysis, robotics, gaming, movie animation, system customization, and dozens of other common domains. Really, it's a powerful, flexible, and what we today call "agile" programming language, that is just as general purpose as C++ or Java.

Python is different, though, because it supports a much more rapid development style. You can do almost everything with Python that you might do in C++ or Java, but you'll probably have to write only 1/3 to 1/5 as much code. For this reason, it's often labeled a "scripting" language, though that mostly refers to its ease of use, not its roles. Python's minimal syntax, dynamic typing, powerful object types, and rich library mean it's easier (and even more enjoyable) to get your work done.

Moreover, unlike some languages in the "scripting" category, Python code is refreshingly readable; that in turn makes it both maintainable and reusable. Because of that, it has attracted scores of people who have struggled with languages like Perl in the past. You can do the same work in Python, but your code will almost automatically be more readable and of higher quality. Readability is crucial in most software projects; because you can more easily read someone else's Python code, it tends to be much more useful at all points of a normal software lifecycle. (I'm not trying to pick on Perl unfairly; I've just seen a lot of ex Perl programmers on the training road who seem to exhibit the same allergic reaction to Perl that I once developed towards C++.)

If Python does have a "niche" I'd say it's popular in any domain where people want to develop software more quickly and less painfully, and be able to take pride in the code of their finished product. That's proven to be a very inclusive domain in practice.

3) Could you tell us how is Python being applied today, especially in web applications? What is the profile of the typical Python user?

Again, I'm not sure there is a "typical" in the Python world; there are far too many users and uses for the language to generalize that way. It's used in just about every conceivable domain in the software field.

Python users run the gamut from novices and hobbyists, to software heavyweights and industry leaders. (It's even used by people like nuclear physicists, who, in my training experiences, are incredibly smart, but often harbor a Fortran-induced knack for writing incredibly bad code—a disease that Python can definitely address!). If there is a common thread among Python users, it's probably their stronger than usual focus on quality. Because the Python language is focused on producing quality software, so too are most Python users.

The web certainly is a big domain for Python. It's used very widely by Google, for example, and YouTube is written almost entirely in Python. The popular BitTorrent P2P file sharing system is coded in Python as well.

These days, most people writing typical web sites with Python seem to be gravitating to the TurboGears and Django web frameworks. These open source Python web frameworks are often described as Python equivalents to Ruby on Rails. Both offer a standard MVC architecture, object/relational database mappers, and more; TurboGears also provides support for Ajax work on the client. To say they are Python's answer to Rails is a bit of an understatement, though; with Python web frameworks you get powerful web development support, but you also get a nicer underlying programming language in Python.

4) Recently, the third edition of your book "Learning Python" was published. How has this book evolved since its first edition back in 1999? Could you describe the structure and approach of this new edition? What can readers expect to learn from it?

There have been two new editions of this book since 1999; both were substantially updated to reflect changes in both Python itself, and in the Python classes I teach.

Learning Python is based directly on my classes. Over the years, the book drew from the classes, and vice versa. According to many readers, the net effect is that reading the book and working through its quiz questions and lab exercises is roughly the same as attending a Python class. You don't get live interaction from a book, of course, but you can reread sections more than once until they sink in.

The current 3rd edition was overhauled to cover Python 2.5, the most recent release, and discusses many anticipated changes in the upcoming 3.0 release. It has fresh coverage of newer language features such as function decorators, context managers, and relative and absolute imports. It also grew new introductory chapters on data types and syntax, which stem from new introductory sessions in classes. In addition, there is expanded coverage of features such as list comprehensions and iterators that have become more widely used tools and best practice since the prior edition.

Also new in the 3rd Edition are summaries and quiz questions at the end of each chapter, called "Brain Builder" sections by O'Reilly's production staff. To be frank, I thought these might be a bit redundant (if not cheesy) when the idea was first proposed, but I was wrong; they are a surprisingly useful review resource that makes the book much stronger as a self-paced learning tool, I think.

5) This is not your first book on Python. How is "Learning Python" different from your previous books, "Programming Python" and the "Python Pocket Reference", in terms of their focus and approach to learning and using Python?

The short story is that Learning Python covers the core Python language; Programming Python is a follow-up text that covers applications programming topics; and the Pocket Reference is a collection of reference materials.

The somewhat longer story is that Learning Python is a tutorial on the core Python language itself, not on its application domains. Its goal is to teach you Python in more depth than most beginners get, and well enough that you will then be able to apply it in whatever domain you work in.

Many people "learn" Python by spending an hour or two going through a tutorial on the web; this works for advanced developers to a certain extent, until they run into strange boundary cases that don't make sense. Learning Python's goal is to teach readers Python's core ideas in enough depth that even the unusual cases will make sense when they crop up. Moreover, it teaches you how to "think Python"—it gives best practice and rule of thumb insights into Python programming that are often difficult to come by with straight reference type materials.

By contrast, Programming Python is a tutorial on common Python application domains—the Web, GUIs, networking, databases, text processing, and so on. It's about what to do with the language after you learn it, and works as a follow-up to Learning Python's language material.

Essentially, Programming Python is to application-level programming topics what Learning Python is to the core language: a gradual teaching tool. It assumes you know the language already, but does not assume you are already an expert in the domains it covers. Because it assumes readers already know Python itself, this is a more advanced text, with more focus on libraries and advanced examples. Unlike the book Python Cookbook, though, this book is a tutorial that starts from square one in each of its topics, and doesn't assume prior proficiency in them. Networking, for example, gradually progresses from basic concepts, to sockets, to client-side tools, to server-side tools, and on to the web and frameworks.

The Pocket Reference is simply reference material. It doesn't teach anything per se, but has proven handy for looking up the fine details once you've mastered the large concepts. Unlike the book Python in a Nutshell, there are no examples or narrative in the Pocket Reference—just quick reference materials to jog your memory in a pinch.

Historical anecdote: really, all 3 of my current books grew out of the 1996 original edition of Programming Python. That book covered both the language and common application domains, and included a reference appendix. As Python grew and new topics sprung up, we realized that a single book wouldn't quite be practical going forward, so we split the core language material off to Learning, the applications topics to Programming, and the reference materials to the Pocket Reference. In retrospect, this turned out to be a great decision, because we were able to focus deeply on each of the topics separately, rather than shallowly together. This split was originally suggested by Frank Willison, my first editor; the commercial success of these books bears out his wisdom in such things.

Having said all that about my own books, though, I want to also say that there are many books available to Python programmers, and books tend to be a fairly subjective experience; have a look for yourself and make up your own mind. (I wouldn't blindly accept an author's recommendation for his or her own book, and neither should you.)

6) What can you tell us about the future of Python? How do you see the language evolving in the coming years?

From a technical standpoint, the impending 3.0 release will be huge, and help spur Python growth in years to come. It's going to add a large set of new features, and take liberty with breaking existing code by abandoning the backward compatibility that Python is normally known for. Essentially, 3.0 (sometimes called Python 3000) is going to clean up things that have been warts on the language for years. Most of the language and its standard library (perhaps 95%) will be the same as it is today, though, especially the large subset of it that most programmers use on a day-to-day basis.

Frankly, after a decade teaching Python to programmers in "the trenches," I've found that many of the new features that appear in Python from time to time appeal mainly to the "bleeding edge" 5% of the Python community. These are interesting and often useful features, of course; it's just that most Python programmers tend to focus more on application libraries they use for their systems, than on arcane language semantics like new-style method resolution order and metaclasses. Not counting some arguably dubious changes like mutating the print statement into a function, for most real-world Python programmers, 3.0 will be a fairly painless migration. Python 3.0 will also come with an auto-conversion script that will update most 2.x code to 3.0 automatically.

From a non-technical standpoint, I think Python is still in a position to continue growing in terms of popularity. Recent evidence suggests that it is already probably the 3rd most widely used programming language (behind C++ and Java, and ahead of Perl and Ruby), but the Python world is still very much on the rise. For example, the "Pycon" Python conference continues to grow in size year over year. This year it had some 1,000 attendees; that's great for a single programming language, and fairly amazing to those of us who remember early conferences in the mid 90s with a few dozen attendees.

Honestly, Python has come so far since 1992 that I've given up predicting how far it will go. As usual, though, its best days are undoubtedly yet to be told.

Thanks for the interview, Mark!

Interview by Paul Zirie