Forward

A Place to Start for the Future Programmer

I guess this all began back in 2002. I was thinking about teaching programming, and what a great language Ruby would be for learning how to program. I mean, we were all excited about Ruby because it was powerful, elegant, and really just fun, but it seemed to me that it would also be a great way to get into programming in the first place.

Unfortunately, there wasn't much Ruby documentation geared for newbies at the time. Some of us in the community were talking about what such a "Ruby for the Nuby" tutorial would need, and more generally, how to teach programming at all. The more I thought about this, the more I had to say (which surprised me a bit). Finally, someone said, "Chris, why don't you just write a tutorial instead of talking about it?" So I did.

And it wasn't very good. I had all these ideas that were good in theory, but the actual task of making a great tutorial for non-programmers was vastly more challenging than I had realized. (I mean, it seemed good to me, but I already knew how to program.)

What saved me was that I made it really easy for people to contact me, and I always tried to help people when they got stuck. When I saw a lot of people getting stuck in one place, I'd rewrite it. It was a lot of work, but it slowly got better and better.

A couple of years later, it was getting pretty good.  So good, in fact, that I was ready to pronounce it finished, and move on to something else. And right about then came an opportunity to turn the tutorial into a book. Since it was already basically done, I figured this would be no problem. I'd just clean up a few spots, add some more exercises, maybe some more examples, a few more chapters, run it by 50 more reviewers...

It took me another year, but now I think it's really really good, mostly because of the hundreds of brave souls who have helped me write it.

What's here on this site is the original tutorial, more or less unchanged since 2004. For the latest and greatest, you'll want to check out the book.

Thoughts For Teachers

There were a few guiding principles that I tried to stick to. I think they make the learning process much smoother; learning to program is hard enough as it is. If you're teaching or guiding someone on the road to hackerdom, these ideas might help you, too.

First, I tried to separate concepts as much as possible, so that the student would only have to learn one concept at a time. This was difficult at first, but a little too easy after I had some practice. Some things must be taught before others, but I was amazed at how little of a precedence hierarchy there really is. Eventually, I just had to pick an order, and I tried to arrange things so that each new section was motivated by the previous ones.

Another principle I've kept in mind is to teach only one way to do something. It's an obvious benefit in a tutorial for people who have never programmed before. For one thing, one way to do something is easier to learn than two. Perhaps the more important benefit, though, is that the fewer things you teach a new programmer, the

more creative and clever they have to be in their programming. Since so much of programming is problem solving, it's crucial to encourage that as much as possible at every stage.

I have tried to piggy-back programming concepts onto concepts the new programmer already has; to present ideas in such a way that their intuition will carry the load, rather than the tutorial. Object-Oriented programming lends itself to this quite well. I was able to begin referring to "objects" and different "kinds of objects" pretty early in the tutorial, slipping those phrases in at the most innocent of moments. I wasn't saying anything like "everything in Ruby is an object," or "numbers and strings are kinds of objects," because these statements really don't mean anything to a new programmer. Instead, I would talk about strings (not "string objects"), and sometimes I would refer to "objects", simply meaning "the things in these programs." The fact that all these things in Ruby are objects made this sort of sneakiness on my part work so well.

Although I wanted to avoid needless OO jargon, I wanted to make sure that, if they did need to learn a word, they learned the right one. (I don't want them to have to learn it twice, right?) So I called them "strings," not "text." Methods needed to be called something, so I called them "methods."

As far as the exercises are concerned, I think I came up with some good ones, but you can never have too many. Honestly, I bet I spent half of my time just trying to come up with fun, interesting exercises. Boring exercises absolutely kill any desire to program, while the perfect exercise creates an itch the new programmer can't help but scratch. In short, you just can't spend too much time coming up with good exercises.

About the Original Tutorial

The pages of the tutorial (and even this page) are generated by a big Ruby program, of course.  All of the code samples were automatically run, and the output shown is the output they generated. I think this is the best, easiest, and certainly the coolest way to make sure that all of the code I present works exactly as I say it does. You don't have to worry that I might have copied the output of one of the examples wrong, or forgotten to test some of the code; it's all been tested.

powered by Ruby

Acknowledgements

Finally, I'd like to thank everyone on the ruby-talk mailing list for their thoughts and encouragement, all of my wonderful reviewers for their help in making the book far better than I could have alone, my dear wife especially for being my main reviewer/tester/guinea-pig/muse, Matz for creating this fabulous language, and the Pragmatic Programmers for telling me about it and, of course, for publishing my book!

If you notice any errors or typos, or have any comments or suggestions or good exercises I could include, please let me know.