Python Chases Monkey

An Introduction to Python Programming

Jason M. Pittman

Preface

Introductory diaglogue

Writing an introduction is fraught with dilemma, connudrum, and misteps. The core problem is deciding just what to write about and planning how the introduction will be simultaneiously entertaining, enticing, and materially relevant. Plus, if you're at all like me, you kind of just what to get on with it. Well, with all that in mind, I think the obvious place to start is with an explanation of how I came up with the title of the book.

The short explaination is I love kung fu and I love Python programming. Thus, it seemed natural to teach Python programming as some pseudo-emulation of Gordon Lui mastering the 36th chamber of Shaolin. If you don't understand that reference, go watch Master Killer. There is a metacontext that is relevant to our upcoming endeavor. I'll come back to that later.

Anyway, I've been fascinated with kung fu ever since I was a kid. I used to stay up late on Friday nights and watch Black Belt Theater, mimicking movements and occasionally even the loud *ki* yells. The best, in my opinion, were the always-present training sequences. You know what I mean - the old, wise master with flowing white hair and beard. The young pupil who learns much more about self-mastery than mastery of kung fu. Yeah, those rock.

For those that have never experienced a Shaw Brothers film, the plots were as predictable as a Nicholas Sparks novel. No matter how hard he fought, our would-be hero would limp away from a devastating defeat, train for a year or two, and then return just powerful enough to defeat the villain. I loved it. I still love it.

The long explanation is that while I may not remember every movie title, I absolutely can recall the long and intricately choreographed fight sequences based on the poetic names the actors called out. Stuff like *dragon whips its tail* or *crane flys to the moon* or whatever. Come on; that's pure literary genius. Moreover, it is actually pedagogical genius.

You see, every student faces down the huge learning curve in memorizing lengthy sequences of information. What those are us that are professional students discover at some point is that encoding information makes learning much more efficient and leads to stronger recall. I'm here to tell you that flowerly phrases such as *tiger claws the dirt* is really just a way to encode a long sequence of physical movements into a coherent, cohesive act.

Looking back, I realize there is a relevant metacontext insofar as learning to develop software requires training, followed by devastating failure, followed by emergent wisdom as a result of even harder training. If you want to go a little deeper, there is also a metacontext relation with Gordon Lui trying the 36th chamber first instead of starting at the beginning. Oddly enough, that is a direct repacking of the Icarus myth. The point is, programming mirrors life and myth in that we often have to learn the difficult lesson of why there is a defined starting point.

This brings me to my next point.

Why I wrote this book

I wrote this book because I think the existing ways to learn programming lack two important traits- namely, unity across examples and relevance throughout exercises. The result is a disjointed, siloed approach to learning.

To be fair, existing programming books are good at showing students *how* elements of programming function. Similarly, recipe books are good at how to make a cake based on a set of steps. However, I don't think those texts are effective at revealing the *why* or *why not* just like a recipe doesn't explain why the procedure results in a cake instead of a pie. This leads programming to be taught as a faux causal procedure: if a program needs to do x, use statement of type y. This is anathema to how we, as humans, most effectively *learn*.

I'll spare you the technical details and summarize my thoughts on learning as: the best way to learn is to rapidly fail. The quicker you iterate through the sequence of guess-try-fail the more information you can incorporate into your subject matter *gestalt*. The more information you incorprate, the richer your experiences will become and the more creative solutions you'll be able to conceive for an increasing variety of problems.

Further, existing programming texts fail to inspire students. The lack of inspiration stems from the habit of *telling* someone what a program is rather than using a metaphor to demonstrate *how* a program can be. The effect is the same as the manner in which mathematics books fail to demonstrate the fun and power of math. In essence, we learn to be bored and uninspired. The antidote to such boredom brings us back to why I think a python chasing a monkey is going to be effective in teaching programming. That is the goal afterall, isn't it?

The goal

The goal of this work is to present Python programming in a way that addresses the two shortcomings I have outlined. Foremost, I use one overarching example throughout the entire book. As well, I provide a transparent view into the thinking associated with *why* specific progrmaming elements are used.

While I cannot guarantee it in all cases, as much as possible, I attempt to avoid telling you what to do and instead use source code snippets to describe what needs to be done. At the end of each chapter, I outline a set of programming exercises, open ended questions, and a project phase to practice your guess-try-fail sequencing.

Lastly, I have attempted to write like I speak: straightfoward and conceptually focused. Take that for what it is worth.

Jason M. Pittman Wallburg, North Carolina December 2020