Reading Guide for Programming with Python for Social Science: 'Specific Vagueness' as a Pedagogical Model

Straight out of the gates, I need to say something really important: *the book is its own reading guide*. What exactly that means will become more apparent as you read the rest of this section, but in practical terms, this means that this section isn't going to be a point-by-point chapter-by-chapter outline of what is contained in the book. Rather, it will be an elaboration on the pedagogical model of the book (which is just fancy teacher-y talk for 'the way the book is meant to help you learn'), to help clarify why things have been done the way they have.

The pedagogical model of the book and the learning to be done with it is drawn from ethnomethodology; a sub-discipline of sociology that is concerned with how people do things in situated context. That's about as specific as I can be without going down a rabbit hole and writing thousands of words on what ethnomethodology is and how it sits in relation to more 'mainstream' sociology. If that seems vague for now, the hope is that things will get increasingly clear as we go.

So, the pedagogical model of the book is grounded in ethnomethodology, but perhaps more specifically, in a particular aspect of it which Harold Garfinkel, the founder of ethnomethodology, reflects in his abiding interest in the local specifics of peoples' activities (learning being one such activity), and which Howard Schwartz, one of Garfinkel's students, calls 'specific vagueness'. What follows is what Howard Schwartz has to say about 'specific vagueness' (which I use rather than Garfinkel's own words only because, coincidentally, Schwartz is writing directly about how students learn to do computer programming):

> 'Beginning computer science students, at a certain university, before writing their first Fortran programme and running it, are given certain advice by their instructor: part of the computer is a device called a compiler which translates Fortran into machine language. The students will be using a compiler program with many error-checking functions. Therefore, if they check the printout when an incorrectly written program is run, it will tell them the specifics of what they did wrong, in many ways. Many, if not most, of these students have never seen or used a computer, compiler, or printout. Both students and instructor understand that the students do not know, in any specific way, what they are being told. The advice is treated as vague, in some specific way. It will become clear by engaging in a certain activity – running the program at the computer centre. At that time, it will become clear both what was told to them, and if it is so. Further, this is the most practical way to both clarify the advice and assess its validity.

> If we construe the advice as instructions for reading the printout, then it might seem that such instructions, in the way that they constrain reading, furnish part of a 'procedure' for looking at printout. But the advice constrains in a peculiar way. Only after the meaning and validity of the advice becomes clear is it available what the constraints were.
>
> In general, we have found specific vagueness to be an extremely prevalent way in which all kinds of instructions are used in the learning of technical skills. As such, it represents a strange but pervasive form of practical enquiry. In such situations, people treat it as the most casual matter that something systematic needs to be done but it will not be until the doing that what needs to be done, and how, will be available.' (Schwartz, 2002: 23 –24)

We don't really need to know what Fortran is (other than it's an old programming language that students working in the 1970s and 1980s, when Schwartz was making his observations, used) or what a compiler is (other than it's something you need to be able to work with, and therefore taught to do, as a Fortran learner). But we can draw several things from this as a pedagogical model, which will help you understand how to read the book and why it's been written in the way it has:

- **Specific vagueness is how we learn *anything* in pretty much all areas of our life**. Ethnomethodology is *really* good at producing descriptions of how things get done in the social world, in all sorts of contexts. Computer programming is one such context that has been looked at as an area of study, but there are also ethnomethodological studies of all sorts of different things like queuing for the bus, working in an air traffic control centre, cooking, playing basketball, making music, how doctors and patients talk in GP clinics, how jury decisions are reached in courts, how scientists working on the Mars Rover mission used their robots to make scientific discoveries... you name it: if it's a social activity, there's likely an ethnomethodological study of it. And learning how to program is one such social activity among many, where formal instructions (such that might be contained within the pages of a book) brush up against the practical problems of putting those instructions to use (e.g. a student attempting to make sense of a programming textbook by following along with the code exercises on their own laptop).
    - As such, our intentions to learn to program are nothing to be daunted by; we learn all sorts of things all the time, and the purpose of this book is to try and teach programming not as something that's inherently new and alien, because it's not. Programming is an everyday activity, which can be taught and learned just like any other everyday activity. So that's what we're going to do here. And given you already know how to learn stuff (otherwise how would you be reading these words?), you're already well-equipped to work your way through this book.

- **Things that are new to us can only have sketchy/indeterminate definitions, until we encounter how those definitions play out in practical work**. Schwartz isn't trying to suggest that learning stuff is completely unproblematic of course – whenever we learn *anything* there will no doubt be times where we get things wrong and get stuck. But what Schwartz *is* saying is that the only way we can learn what programming is and how to do it is through encountering those problems *in situ* (which is to say, in the context of trying to apply sanitised formal instructions to the messy 'rough ground' (Wittgenstein, 2009: §107) of real world practices).
    - As such, it's no use to *only* read the book (or chapters of it) without engaging with the materials practically as suggested. That's not going to work. If you brush through bits of the book, you *will* get a small sense of what programming is about and how to go about doing it, but you won't be a PaSS (Programming-as-Social-Scientist) practitioner. What you get by engaging in the practical work that the book provides an opportunity for you to undertake, is a sense of how to make programming work for *you*: this is not just to do with fitting Python into a specifically social-scientific research context (i.e. questions about how to do programming in ways that respond to social problems, how to talk theoretically about research involving programming, etc), but also just the nuts and bolts of getting Python to work on your machine and applying it to different types of task (i.e. questions about how to install add-on libraries on a Windows laptop, how to set up a Twitter developer account for data collection, etc).
    - Importantly, there is no way I can give a full set of formal instructions to follow here. For instance, your computers and the files on them will inevitably be structured differently to mine, so I can't tell you the exact location of where to find the relevant area to install add-on libraries from/to. Neither can I give you a full set of instructions for setting up a Twitter account with developer access; Twitter changes the way this works periodically, so it's pointless me trying to write such a set of instructions since I know they'll be out of date soon in due course. However, what I *can* do is set you off in the right direction by telling you how *I've* done things in such a way that you'll be able to see where your specific context differs and what you might need to do differently to get things to work for *you*. You'll spend a lot of time in this book doing exactly this: following along with me, applying what I've said to your own specific context, and likely encountering (and solving) your own specific problems as you go. But, that's exactly the kind of skill required, because...

- **Learning to do something is learning the *method* to do that thing**. In Python programming, as with anything, there can be some bits of knowledge where it's just necessary to almost learn it by rote – e.g. 'you declare a function by writing "def" at the beginning', that kind of thing. But in Python programming, as with anything, rote learning only gets you so far; in order to apply Python creatively and critically (which we'll want to do, as social scientists), we need to be able to think about how we use the knowledge we have to go beyond just copying techniques from a book or from other people. And the materials in this book have been designed to help you do exactly that.
    - So this is a really valuable way of learning that is going to stand you in good stead long after you've finished with the book. Learning the methods to do the various things covered in the book is not just going to help you know how to do that thing, but will also give you the knowledge and skills you need to go on and apply those things in other contexts, creatively and critically.
    - However, the trade-off is that there are no shortcuts here; it'll only cause problems if you gloss over chapters or exercises (inasmuch as you won't then be able to solve the practical problems the chapters and exercises are designed to put you in touch with), so spend as much time as you need to on each bit to make sure you've got what you need from it.

So, that's what the pedagogical model of 'specific vagueness' is about, and now we can point more directly to how it informs what you're going to see in this book. The 'formal instructions' I give in the book are to do one specific (and sometimes quite small) task in Python, in one particular specific context. You can follow along with these – type the code out, run it for yourself – to get a handle on what the work of programming involves. But this is only ever a jumping-off point for your learning – the more opportunities you give yourself to apply these concepts in other specific practical contexts outside of the scope of this book, the more you learn.

Following along with the 'instructions' in the book will, by design, bring you into contact with a range of practical problems of programming – things like working out what comments are for, how to read error messages, what the difference is between 'global' and 'local' variables, how the structure of logical conditions (i.e. 'IF/ELSE' statements) can affect how your code runs, and so on. These are *not* just trivialities to be overcome, but exactly part of what you need to be able to cope with to learn programming. To help you do that, the book is written and structured to provide you opportunities to encounter those practical problems. It is also written in such a way as to help you *recognise* those problems, give you tips on how to *think about* those problems, and provide support as you figure out

what you can do to solve these kinds of problems. The book will, therefore, feature regular reminders as to how you should be engaging with the various elements contained within; this is to say that learning how to read this book doubles up as also learning how to program, since the process of actually doing programming has been laid out as transparently as I could make it. As above, learning to do programming involves learning how to learn about programming, and this book is intended to walk you through both simultaneously.

Though it is a textbook (and textbooks are perhaps conventionally not read from start to finish, but dipped in and out of), the book has been designed so that you will encounter these practical problems in a specific order, such that insights accumulate and build on earlier learning. It is therefore recommended that for your first reading of the book, you do work through it from page 1 to the end (including the more text-based chapters: don't skip those!). It might seem like more work to do so but it's *necessary* work and, paradoxically, skipping through the book is more likely to mean you have a harder time (and spend longer) trying to figure out what things mean. So, read the book start-to-finish on your first read, but once you've done that learning the individual chapters will also stand as reference material to refresh yourself with in the more standard textbook-y fashion.

It will also likely bring you into contact with mistakes you make yourself. For instance, if you have a habit of missing out commas to separate entries in a list, you'll be confronted with this time and time again as you're working through the book's exercises, which means you'll have plenty of opportunities to see what you're doing wrong, and plenty of opportunities to do it right. Doing things wrong is *absolutely not a bad thing* here. Making mistakes, and figuring out how to correct them, accounts for quite a lot of what programming is about! So the more chances you have to do that, the better a programmer you will be. Furthermore, copying and pasting is 'cheating' in this context. Constructing code for yourself, perhaps inevitably getting things wrong, correcting those things, and ultimately getting a script to work; *that's* what counts as success here.

So, as you read the book and work your way through it, you'll hopefully start to see more clearly how this book is its own reading guide; in order to learn how to do programming in Python, you need to learn *how to learn to do programming in Python*, and the only way to do that is to jump right in and try it out. But really, any summary description that I might write in a concluding paragraph is going to be inadequate, since it's precisely the point of 'specific vagueness' as a pedagogical model that a full description of this process is a conceptual impossibility; the best way to see what all of the above means is to open the book at page 1, and see for yourself. Have fun with it!

References

Schwartz, H. (2002) Data: Who needs it?, *Ethnographic Studies* 7: 7 –32.

Wittgenstein, L. (2009) *Philosophical Investigations* (trans. Anscombe, G. E. M., Hacker, P. M. S. and Schulte, J.). Chichester, UK: Wiley-Blackwell.