I Wrote a Book—To Teach the Wolfram Language

December 8, 2015

An Elementary Introduction to the Wolfram Language is available in print, free on the web, etc.

I wasn’t sure if I was ever going to write another book. My last book—A New Kind of Science—took me more than a decade of intensely focused work, and is the largest personal project I’ve ever done.

But a little while ago, I realized there was another book I had to write: a book that would introduce people with no knowledge of programming to the Wolfram Language and the kind of computational thinking it allows.

The result is An Elementary Introduction to the Wolfram Language, published today in print, free on the web, etc.
The goal of the book is to take people from zero to the point where they know enough about the Wolfram Language that they can routinely use it to create programs for things they want to do. And when I say “zero”, I really mean “zero”. This is a book for everyone. It doesn’t assume any knowledge of programming, or math (beyond basic arithmetic), or anything else. It just starts from scratch and explains things. I’ve tried to make it appropriate for both adults and kids. I think it’ll work for typical kids aged about 12 and up.

In the past, a book like this would have been inconceivable. The necessary underlying technology just didn’t exist. Serious programming was always difficult, and there wasn’t a good way to connect with real-world concepts. But now we have the Wolfram Language. It’s taken three decades. But now we’ve built in enough knowledge and automated enough of the process of programming that it’s actually realistic to take almost anyone from zero to the frontiers of what can be achieved with computation.
It's a Conversation

In essence, the book brings the reader into a conversation with the computer. There are two great things about the Wolfram Language that make this really work. First, that the language is symbolic, so that anything one's dealing with—a color, an image, a graph, whatever—can be right there in the dialog. And second, that the language can be purely functional, so that everything is stateless, and every input can be self contained.

Make a list of colors:

    In[1] := (Red, green, Blue, Purple, Orange, Black)
    Out[1] := {Red, Green, Blue, Purple, Orange, Black}

You can do operations on colors. ColorNegate negates a color, giving the complementary color. Blend blends a list of colors together.

Negating the color yellow gives blue:

    In[2] := ColorNegate[Yellow]

Here's the result of blending yellow, pink and green:

    In[3] := Blend[{Yellow, Pink, Green}]
It's also very important that the Wolfram Language has **built-in knowledge** that lets one immediately compute with real-world things.

Where to Start? OK, but where should one start? The very first page is about arithmetic—just because that's a place where everyone can see that a computation is actually happening:
3/18/2018

There's a section called Vocabulary because that's what it is: one's learning some “words” in the Wolfram Language. Then there are exercises, which I’ll talk about soon.

OK, but once one’s done arithmetic, where should one go next? What I decided to do was to go immediately to the idea of functions—and to first introduce them in terms of arithmetic. The advantage of this is that while the concept of a function may be new, the operation it’s doing (namely arithmetic) is familiar.
And once one’s understood the function `Plus`, one can immediately go to functions like `Max` that don’t have special input forms. What `Max` does isn’t that exciting, though. So as a slightly more exciting function, what I introduce next is `RandomInteger`—which people often like to run over and over again, to see what it produces.

OK, so what next? The obvious answer is that we have to introduce lists. But what should one do with lists? Doing something like picking elements out of them isn’t terribly exciting, and it’s hard immediately to see why it’s important. So instead what I decided was to make the very first function I show for lists be `ListPlot`. It’s nice to start getting in the idea of visualization—and it’s also a good example of how one can type in a tiny piece of code, and get something bigger and more interesting out.
Actually, the best extremely simple example of that is **Range**, which I also show at this point. **Range** is a great way to show the computer actually computing something, with a result that’s easy to understand.

But OK, so now we want to reinforce the idea of functions, and functions working together. The function **Reverse** isn’t incredibly common in practice, but it’s very easy to understand, so I introduce it next, followed by **Join**.
What’s nice then is that between `Reverse`, `Range` and `Join` we have a little microlanguage that’s completely self-contained, but lets us do a variety of computations. And, of course, whatever computations one does, one can immediately see the results, either symbolically or visually.
The next couple of sections talk about displaying and operating on lists, reinforcing what’s already been said, and introducing a variety of functions that are useful in practice. Then it’s on to Table—a very common and powerful function, that in effect packages up a lot of what might otherwise need explicit loops and so on.

I start with trivial versions of Table, without any iteration variable. I take it for granted (as people who don’t know “better” do!) that Table can produce a list of graphics just like it can produce a list of numbers. (Of course, the fact that it can do this is a consequence of the fundamentally symbolic character of the Wolfram Language.)

The next big step is to introduce a variable into Table. I thought a lot about how to do this, and decided that the best thing to show first is the purely symbolic version. After all, we’ve already introduced functions, and with the symbolic version, one can immediately see where the variable goes. But now that we’ve got Table with variables, we can really go to town and start doing what people will think of as “real computations”.

6 | Making Tables

We’ve seen a few ways to make lists in the Wolfram Language. You can just type them in. You can use Range. And you can use functions like IntegerDigits. One of the most common and flexible ways to make lists is with the function Table.

In its very simplest form, Table makes a list with a single element repeated some specified number of times.

```
Table[0, 2]
```

This makes a list with 2 repeated 0 times:

```
{0, 0}
```

You can repeat the zero:

```
Table[0, {4}]
```

```
{0, 0, 0, 0}
```

Or, actually, anything here’s a list of four identical elements:

```
Table[0, {4}, {4}]
```

```
{{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}
```

But what if we want to make a table where the elements aren’t identical? We can do that by introducing a variable, and then filling in that variable.

Essentially we’re making a list where the variable goes up to 3:

```
Table[0, {4}, {x}]
```

```
{{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}
```

Here’s how this works. To make the first element of the set, it’s taken to be 3, so 0 is 0. To make the second element, it’s taken to be 2, so 0 + 0 is 0, and so on. It’s called a variable because it’s varying as we make the different elements of the list.

```
Table[0, x, {4}]
```

```
{{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}
```

http://blog.stephenwolfram.com/2015/12/i-wrote-a-book-to-teach-the-wolfram-language/
The Arc of the Book  In the first few sections of the book, the raw material for our computations is basically numbers and lists. What I wanted to do next was to show that there are other things to compute with. I chose colors as the first example. Colors are good because (a) everyone knows what they are, (b) you can actually compute with them and (c) they make colorful output (!).
After colors we're ready for some graphics. I haven't talked about coordinates yet, so I can only show individual graphical objects, without placement information.

Now we're all set up for something "advanced": interactive manipulation. It's pretty much like Table, except that one gets out a complete interactive user interface. And
since we’ve introduced graphics, those can be part of the interface. People have seen interactive interfaces in lots of consumer software. My experience is that they’re pretty excited to be able to create them from scratch themselves.

The next, perhaps surprising thing I introduce in the book is image processing. Yes, there’s a lot of sophisticated computation behind image processing. But in the Wolfram Language that’s all internal. And what people see are just functions—like Blur and ColorNegate—whose purposes are easy to understand.

It’s also nice that people—especially kids—can compute with images they take, or drag in. And this is actually the first example in the book where there’s rich data coming into a computation from outside. (I needed a sample image for the section, so, yes, I just snapped one right there—of me working on the book.)
Next I talk about strings and text. String operations on their own are pretty dry. But in the Wolfram Language there’s lots of interesting stuff that’s easy to do with them—like visualizing word clouds from Wikipedia, or looking at common words in different languages.
Next I cover sound, and talk about how to generate sequences of musical notes. In the printed book you can’t hear them, of course, though the little score icons give some sense of what’s there.

12 | Sound

In the Wolfram Language, sound works a little like graphics, except that instead of looking things up digitally, one has named sounds. Thus for example it does not actually play sounds. If you don’t say otherwise, the Wolfram Language will make the notes sound as if they were played on a piano.

```
Sound[SoundNote["C"]]
```

You can specify a sequence of notes by giving them in a list.

```
Sound[{SoundNote["C"], SoundNote["C"], SoundNote["D"]}]
```

Instead of playing notes of sound, you can give a number to specify their pitch. Middle C is 440 Hz. Each one a comma above middle C goes up by 1. Middle C is 440 Hz, one above middle C is 442 Hz, and so on.

```
Sound[{12, 12, 12}]
```

Use Table to generate a sequence of notes:

```
Sound[Table[SoundNote[i], {i, 5}]]
```

One might wonder, “Why not talk about sound right after graphics?” Well, first of all, I thought it wasn’t bad to mix things up a bit, to help keep the flow interesting. But more than that, there’s a certain chain of dependencies between different areas. For example, the names of musical notes are specified as strings—so one has to have talked about strings before musical notes.

Next it’s “Arrays, or Lists of Lists”. Then it’s “Coordinates and Graphics”. At first, I worried that coordinates were too “mathy”. But particularly after one’s seen arrays, it’s not so difficult to understand coordinates. And once one’s got the idea of 2D coordinates, it’s easy to go to 3D.
By this point in the book, people already know how to do some useful and real things with the Wolfram Language. So I made the next section a kind of interlude—a meta-section that gives a sense of the overall scope of the Wolfram Language, and also shows how to find information on specific topics and functions.
Now that people have seen a bit about abstract computation, it’s time to talk about real-world data, and to show how to access the vast amount of data that the Wolfram Language shares with Wolfram|Alpha.

Lots of real-world data involves units—so the next section is devoted to working with units. Once that’s done, we can talk about geocomputation: things like finding distances on the Earth, and drawing maps.
After that I talk about dates and times. One might think this wouldn’t be an interesting or useful topic. But it’s actually a really good example of real-world computation, and it’s also something one uses all over the place.
cover these ideas. And the next section—on options—covers one such simple idea that’s widely used in practice.

After covering options, we’re set to talk about something that’s often viewed as a quite advanced topic: graphs and networks. But my experience is that in modern times, people have seen enough graphs and networks in their everyday life that they don’t have much trouble understanding them in the Wolfram Language. Of course, it helps a lot that the language can manipulate them directly, as just another example of symbolic objects.
After graphs and networks, we’re ready for another seemingly very advanced topic: **machine learning**. But even though the internal algorithms for machine learning are complicated, the actual functions that do it in the Wolfram Language are perfectly easy to understand. And what’s nice is that by doing a bunch of examples with them, one can start to get pretty good high-level intuition about the core ideas of machine learning.
Throughout the book, I try to keep things as simple as possible. But sometimes that means I have to go back for a deeper view of a topic I’ve already covered. “More about Numbers” and “More Forms of Visualization” are two examples of doing this—covering things that would have gotten in the way when numbers and visualization were first introduced, but that need to be said to get a full understanding of these areas.

**Functional Programming**  The next few sections tackle the important and incredibly powerful topic of functional programming. In the past, functional programming tended to be viewed as a sophisticated topic—and certainly not something to teach people who are first learning about programming. But I think in the Wolfram Language the picture has changed—and it’s now possible to explain functional programming in a way that people will find easy to understand. I start by just talking more abstractly about the process of applying a function.

The big thing this does is set me up to talk about **pure anonymous functions**. In principle I could have talked about these much sooner, but I think it’s important for people to have seen many different kinds of examples of how functions are used in general—because that’s what’s needed to motivate pure functions.
The next section is where some of the real power of functional programming starts to shine through. In the abstract, functions like `NestList` and `NestGraph` sound pretty complicated and abstract. But by this point in the book, we’ve covered enough of the Wolfram Language that there are plenty of concrete examples to give—that are quite easy to understand.
The next several sections cover areas of the language that are unlocked as soon as one understands pure functions. There are lots of powerful programming techniques that emerge from a smaller number of ideas.

After functional programming, the next big topics are patterns and pattern-based programming. I could have chosen to talk about patterns earlier in the book, but they weren’t really needed until now.
What makes patterns so powerful in the Wolfram Language is something much more fundamental: the uniform structure of everything in the language, based on symbolic expressions. If I were writing a formal specification of the Wolfram Language, I would start with symbolic expressions. And I might do the same if I were writing a book for theoretical computer scientists or pure mathematicians.

It’s not that symbolic expressions are a difficult concept to understand. It’s just that without seeing how things actually work in practice in the Wolfram Language, it’s difficult to motivate abstractly studying them. But now it makes sense to talk about them, not least because they let one see the full power of what’s possible with patterns.

**The Whole Stack**  At this point in the book, we’re getting ready to see how to actually deploy things like web apps. There are a few more pieces to put in place to get there. I talk about associations—and then I talk about natural language understanding. Internally, the way natural language understanding works is complex. But at the level of the Wolfram Language, it’s easy to use—though to see how to connect it into things, it’s helpful to know about pure functions.
OK, so now everything is ready to talk about deploying things to the web. And at this point, people will be able to start creating useful, practical pieces of software that they can share with the world.

It’s taken 220 pages or so. But to me that’s an amazingly small number of pages to go from zero to what are essentially professional-grade web apps. If we’d just been talking
about some very specific kind of app, it wouldn’t be so impressive. But we’re talking about extremely general kinds of apps, that do pretty much any kind of computation.

**Assigning Values to Variables**  If you open a book about a traditional programming language like C++ or Java, one of the first things you’re likely to see is a discussion of assigning values to variables. But in my book I don’t do this until Section 38. At some level, this might seem bizarre—but it really isn’t. Because in the Wolfram Language you can do an amazing amount—including for example deploying a complete web app—without ever needing to assign a value to a variable.

And this is actually one of the reasons why it’s so easy to learn the Wolfram Language. Because if you don’t assign values to variables, every piece of code in the language stands alone, and will do the same thing whenever it’s run. But as soon as you’re assigning values to variables, there’s hidden state, and your code will do different things depending on what values variables happen to have.

Still, having talked about assigning values to variables—as well as about patterns—we’re ready to talk about defining your own functions, which is the way to build up more and more sophisticated functionality in the Wolfram Language.
At this point, you’re pretty much set in terms of the basic concepts of the Wolfram Language. But the last few sections of the book cover some important practical extensions. There’s a section on string patterns and templates. There’s a section on storing things, locally and in the cloud. There’s a section on importing and exporting. And there’s a section on datasets. Not everyone who uses the Wolfram Language will ever need datasets, but when you’re dealing with large amounts of structured data they’re very useful. And they provide an interesting example that makes use of many different ideas from the Wolfram Language.

**Essay Sections** At the end of the book, I have what are basically essay sections: about writing good code, about debugging and about being a programmer. My goal in these sections is to build on the way of thinking that I hope people have developed from reading the rest of the book, and then to communicate some more abstract principles.
Structuring the Presentation  I said at the beginning of this post that the book is essentially written as a conversation. In almost every section, I found it convenient to add two additional parts: Q&A and Tech Notes. The goal of Q&A is to have a place to answer obvious questions people might have, without distracting from the main narrative.
There are several different types of questions. Some are about extensions to the functionality that’s been discussed. Some are about the background to it. And some are questions (“What does ‘raised to the power’ mean?”) that will be trivial to some readers but not to others.

In addition to Q&A, I found it useful to include what I call Tech Notes. Their goal is to add technical information—and to help people who already have sophisticated technical knowledge in some particular area to connect it to what they’re reading in this book.

**Exercises** Another part of most sections is a collection of exercises. The vast majority are basically of the form “write a piece of code to do X”—though a few are instead “find a simpler version of this piece of code”.

There are answers to all the exercises in the printed book at the back—and in the web version there are additional exercises. Of course, the answers that are given are just possible answers—and they’re almost never the only possible answers.
Writing the exercises was an interesting experience for me, that was actually quite important in my thinking about topics like how to talk to AIs. Because what most of the exercises effectively say is, “Take this description that’s written in English, and turn it into Wolfram Language code.” If what one’s doing is simple enough, then English works quite well as a description language. But when what one’s doing gets more complicated, English doesn’t do so well. And by later in the book, I was often finding it much easier to write the Wolfram Language answer for an exercise than to create the actual exercise in English.

In a sense this is very satisfying, because it means we really need the Wolfram Language to be able to express ideas. Some things we can express easily in English—and eventually expect Wolfram|Alpha to be able to understand. But there’s plenty that requires the greater structure and precision of the Wolfram Language.

A Book? At some level it might seem odd in this day and age to be writing a book that can actually be printed on paper, rather than creating some more flexible online structure. But what I’ve found is that the concept of a book is very useful. Yes, one can have a website where one can reach lots of information by following links. But when people are trying to systematically learn a subject, I think it’s good to have a definite, finite container of information, where there’s an expectation of digesting it sequentially, and where one can readily see the overall structure.

That’s not to say that it’s not useful to have the book online. Right now the book is available as a website, and for many purposes this web version works very well. But somewhat to my surprise, I still find the physical book, with its definite pagination and browsable pages, better for many things.

Of course, if you’re going to learn the Wolfram Language, you actually need to run it. So even if you’re using a physical book, it’s best to to have a computer (or tablet) by your side—so you can try the examples, do the exercises, etc. You can do this.
What’s Not in the Book  I call the book an "elementary introduction". And that’s what it is. It’s not a complete book about the Wolfram Language—far from it. It’s intended to be a basic introduction that gets people to the point where they can start writing useful programs. It covers a lot of the core principles of the language—but only a small fraction of the very large number of specific areas of functionality.

Generally I tried to include areas that are either very commonly encountered in practice, or easy for people to understand without external knowledge—and good for illuminating principles. I’m very happy with the sequence of areas I was able to cover—but another book could certainly pick quite different ones.

Of course, I was a little disappointed to have to leave out all sorts of amazing things that the Wolfram Language can do. And at the end of the book I decided to include a short section that gives a taste of what I wasn’t able to talk about.

Some Backstory  I see my new book as part of the effort to launch the Wolfram Language. And back in 1988, when we first launched Mathematica, I wrote a book for that, too. But it was a different kind of book: it was a book that was intended to provide a complete tutorial introduction and reference guide to the whole system. The first edition was 767 pages. But by the 5th edition a decade later, the book had grown to 1488 pages. And at that point we decided a book just wasn’t the correct way to deliver the information—and we built a whole online system instead.
It’s just as well we did that, because it allowed us to greatly expand the depth of coverage, particularly in terms of examples. And of course the actual software system grew a lot—with the result that today the full Documentation Center contains more than 50,000 pages of content.

Many people have told me that they liked the original Mathematica book—and particularly the fact that it was short enough to realistically read from cover to cover. My goal with An Elementary Introduction to the Wolfram Language was again to have a book that’s short enough that people can actually read all of it.

Looking at the book, it’s interesting to see how much of it is about things that simply didn’t exist in the Wolfram Language—or Mathematica—until very recently. Of course it’s satisfying to me to see that things we’re adding now are important enough to make it into an elementary introduction. But it also means that even people who’ve known parts of the Wolfram Language through Mathematica for many years should find the book interesting to read.

Why Me? I’ve thought for a while that there should be a book like the one I’ve now written. And obviously there are plenty of people who know the Wolfram Language well and could in principle have written an introduction to it. But I’m happy that I’ve been the one to write this book. It’s reduced my productivity on other things—like writing blogs—for a little while. But it’s been a fascinating experience.

It’s been a bit like being back hundreds of years and asking, “How should one approach explaining math to people?” And working out that first one should talk about arithmetic, then algebra, and so on. Well, now we have to do the same kind of thing for computational thinking. And I see the book as a first effort at communicating the tools of computational thinking to a broad range of people.

It’s been fun to write. I hope people find it fun to read—and that they use what they learn from it to create amazing things with the Wolfram Language.