Quant Perspectives

Six Great Reasons *You* Should Learn to Code

Computer code is the future – and it's also the present. To improve now and be much more valuable in the future, learn to code! The benefits are immense.

By Joe Pimbley

It sometimes feels like our world consists of "coders" – also known as "developers" or "programmers" – and, well, everyone else. If you don't code, you don't code. Neither do you speak the language of the coders or understand their ways.

Outsiders can and probably should feel excluded. If you don't code by the end of your undergraduate years, the conventional wisdom is "that ship has sailed." The common view is that the barrier to entry is too high and the learning curve is too steep.

Don't surrender to these thoughts! My view is that learning to code at any age or level of professional seniority has great advantages. It's also much easier than many believe.

My colleague, Rob Chang, and I presented a "Learning Lab" at the (March 2017) *GARP 18th Annual Risk Management Convention* in New York in which we taught Visual Basic and Python code at the entry level. This webpage provides the Excel VBA and Python files of our tutorial – as well as two videos and an earlier GARP column describing *Monte Carlo* code.

Now, let's consider six reasons why *you* should learn to code (also available in a video version we prepared for this article): (**Joe, I think we will be able to maintain readers' interest better by revealing the reasons one-by-one, rather than all at once. That's why I'm suggesting we remove the group of bullets.)**

1. You will be a better model user.

Most likely, you are a user of models for risk management, valuation, stress testing and other purposes. Learning to write the underlying code yourself – even in greatly simplified form – will give you a deeper understanding of the sophisticated models of other developers you use.

When you write your own code, for example, you make mistakes. You see the mistakes when you run your code and learn how to fix (or "debug") the code. You develop the ingrained habit of *questioning all model results* you see!

Don't misunderstand. Some model calculations are correct. That is, the code sometimes properly executes the intent of the model. But *all* code has "bugs" in its first stages. When you write your own simple code and find the bugs and succeed in the debugging, you will be a much better and wiser user of models.

2. You will be a better judge of code and coders.

As you learn to write elementary code, you will become a better judge of the quality of both existing code and the developers. At the level of the user interface (UI), you will gain appreciation for simplicity and "absence of crashes."

Creating a UI that is intuitive and that easily handles missing or mis-typed input information can be painfully difficult to code! Only when you begin coding yourself will this quality feature be evident.

One of my favorite activities is to ask a developer to provide the "source code." This is the actual list of statements and instructions in the specific language, such as Java or C++. You will get a wide range of reactions!

First, let me clarify that source code is proprietary information. You would not, and should not, ask for the source code of a friend at a different firm. But if you're a user colleague, or a manager, of a model developer, then it's not unreasonable to ask to read the source code.

An excellent coder will provide this code with no delay, and may even want to tell you all the clever ideas and methods he/she employs. But many developers will be hesitant to share their source code. They may tell you they need to “clean it up,” and will promise to send the code in a few days or a week. Moreover, there will be some coders who simply refuse to divulge their code.

It is a fascinating professional and “people” issue to request source code. You learn something about the person and the likely quality of code just by making the request.

If you do get the code, then you have the opportunity to learn the code quality. Even if your coding skills are at an elementary level, you will understand what you see much better than if your “code level” is zero.

Well-written code has numerous “comments” that briefly explain the function of each block or section. Such code has both visual and logical organization that is apparent without understanding details. In contrast, poorly-written code typically has few or no comments and just looks messy and chaotic.

Of course, what we call “well-written” code may still have bugs. But the corollary of “poorly-written code having no bugs” is far less likely.

Apart from forming an opinion of the code and the coder, reading the code of an expert is the best way to learn. As you run the model, try to trace through within the code what the model is doing.

Do something simple like adding a print statement to the code. Run this revised code and see if your printed message works as planned. Then move on to more important features. There is immense self-study in learning to code!

3. You may be the best person to write the code.

Our advice above is to work with the code of an expert colleague as one learning technique. That is, instead of creating a program beginning with a blank slate, start with a model you use. From the user perspective, you understand the model and what it should do. Your goal now is to learn the code part – which is the behind-the-scenes action.

As a related suggestion, choose a model or problem in which you either are a subject matter expert or otherwise have great interest. That will make this code tutorial meaningful to you.

There are many good online resources for learning to code. One disadvantage of all of them is that they must choose an irrelevant example problem.

Examples are good, but irrelevant examples are not motivating. Imagine one of your jobs is to run bank stress testing models. To learn the code behind the models, request the source code and make simple changes (double all volatility parameters!) to see the impact.

As you learn, you'll eventually reach situations in which you are the best person to code the next model. It's not that you'll be the best coder in your vicinity. Rather, it's the combination of your expertise in the topic and your new facility with code.

Your strength is your existing knowledge and command of your field. It is easier for you to learn to code than it is for the developer to learn all the business and analytic details relevant to your problem.

4. You may be creating your next career.

The future is code. Everything you learn now about writing code will help in that future. Even if you don't become a developer yourself, you'll manage the transition better to "smart contracts" and other innovations.

5. You will be surprised by what you learn.

A philosophical quote I admire greatly is Goethe's "We see only what we know." The greater your knowledge range – especially outside of your expertise – the more you will see in the world. In my mind, this is the single greatest benefit of continuous learning.

You do not really "see" web pages until you learn some HTML (an acronym for "hypertext markup language"). First, let me give my opinion that HTML is not truly a "language" in the sense of most languages. It's different – it's really just a set of page layout instructions. There’s nothing wrong with that. That’s just one thing I learned.

More importantly, HTML is obsessively focused on rectangles! When you study HTML and then look at web pages, you see rectangles that hold text, rectangles that hold images, rectangles that hold tables and lists, even rectangles that hold other rectangles.

Similarly, as you learn to code, you will notice there are bugs everywhere! When your smartphone or computer operating system next asks you to install a software update, read the description. In addition to claimed improvements, you’ll see that the update is fixing broken stuff! They’re still fixing bugs in version 10.2, or whatever.

When your browser next hangs and you delete the session and re-start, you’ll appreciate the software has failed – another bug! In properly functioning code, the user does not need to delete and re-start.

I don’t know what surprising elements *you* will learn as you begin to code, but neither do you!

6. You may love coding!

As we discussed earlier, find a problem that motivates you. Once you discover that writing your own code teaches you an aspect of this problem you hadn’t previously known, you’ll feel that great sensation of something clicking in your brain. For this reason alone, you’ll fold coding into your professional skill set.

But coding is more than that. It’s all puzzles. You know outside of the code what calculation to do or problem to solve or question to answer – but your programming is not at all a literal translation of a human thought process. Rather, you must learn the rules and methods of the computer language and then figure out how that (fairly simple) language can do your calculation (or solve your problem or answer your question).

Full-time coding is conquering brain quizzes all day. You may find that you love it!

Joe Pimbley (FRM) is a financial consultant in his role as Principal of Maxwell Consulting, LLC. His expertise includes enterprise risk management, structured products, derivatives, investment underwriting, training and quantitative modeling. Find Joe's archive of previous GARP columns here and his collection of short code projects here.