Chapter 6

Learning to Code

Whenever I speak on library code issues, one of the first questions I get is, "How can I learn to code?" If that was your question, this chapter is for you. I'll discuss respondents' recommendations for learning strategies and resources. I'll also cover the various forms of workplace support librarians have received in learning to code so that you know what to ask of your manager, or what to provide if you are a manager.

Learning Strategies and Resources That Coders Recommend

I asked survey respondents what they would recommend to people who'd like to learn to code. The recurring themes were these:

- find a project
- rely on Google and existing code
- write documentation
- persevere
- find a mentor

Of these, *finding a project* is the most important. It doesn't matter if it's for work or for fun, though it will be easier to get professional development support for work projects; it just has to be important to you. Having a goal you're committed to will help you persevere through the inevitable challenges (see below). It will give you a sense of accomplishment when you make progress; it may even have real-world impact, which is tremendously motivational for many coders. It can also provide natural answers to questions like "What programming language should I learn?" and "What do I need to learn next?"

What sort of project? You may already have one in mind, in which case, start there! If not, automate a repetitive task, simplify a bothersome workflow, or improve some element of user experience. Or, of course, take on one of the projects in this report! Most of them can be accomplished in under a hundred lines of code; you'll need a solid grasp of programming fundamentals, but you don't need a deep grounding in computer science or years of experience. Write one from scratch, rewrite one in your preferred language, or modify one to work better for you; the scripts in this report are intended to be a springboard for you. Whatever you choose, make it as small as possible (or break it down into small parts) so it doesn't get too overwhelming, and feel free to incorporate working code snippets you find online. The sooner you can get something interesting working, the sooner you'll feel rewarded and capable.

This brings us to the second piece of advice, *rely on Google and existing code*. Modifying existing code is not cheating! There's a good chance someone else has already written code to do most of what you want; the ability to read and edit others' code can get you a long way, even if you never write your own programs from scratch. Even experienced programmers regularly look up syntax details and copy and paste code snippets from around the web. Googling for something like "[programming language] [problem keyword] example" will often turn up helpful code samples and StackOverflow advice. Spending some quality time browsing library coders' GitHub repositories can yield lots of useful code and inspiration, too. The Code4Lib wiki page "Libraries Sharing Code" is a good starting place. Many of the people cited in this report have GitHub repositories as well.

StackOverflow
http://stackoverflow.com

Libraries Sharing Code
http://wiki.code4lib.org/Libraries_Sharing_Code

Not familiar with GitHub? You don't need an account to browse and download code. However, it's more useful once you have an account so that you can fork repositories (i.e., make your own copy to edit) and master a few basic commands. The LITA Library Code Year Interest Group has a hands-on tutorial available.

Learn GitHub tutorial
https://github.com/LibraryCodeYearIG/Codeyear-IG-Github-Project

Google, StackOverflow, and (to a lesser extent) GitHub work as learning tools because people have invested time in *documentation*. Pay it forward! Writing up your own learning process can be helpful to those who come after you—notably including yourself in six months, when you've forgotten everything you were thinking today. Organizing your thoughts well enough to write them is a good self-teaching tool. Additionally, many open-source projects want help with documentation as well as code, and this can be an easier route than code to begin contributing. Read the project guidelines, look for a bug tracker with open documentation bugs, and make things better while your memory is fresh. Finally, writing documentation increases the chances that others will build on your work; seeing others succeed because of your work can be motivational and rewarding.

Step four: *persevere*. Learning to code is hard! You must devote a lot of time to it. Also, you'll make mistakes, and some of them will be hard to debug. Beginners often think this means they don't have the aptitude, but they're wrong; coders at all levels constantly run into challenging bugs. As Kate Roy says, "There is no mastery, there is no final level. The anxiety of feeling lost and stupid is not something you learn to conquer, but something you learn to live with."[1] Or, as Cecily Carver notes, in an outstanding Medium article on what she wishes she'd known as a new coder, "I've found that a big difference between new coders and experienced coders is faith: faith that things are going wrong for a logical and discoverable reason, faith that problems are fixable, faith that there is a way to accomplish the goal."[2]

People don't talk enough about emotion in learning to code. They talk about languages and tools and MOOCs and books, but not about feelings: about the intense ways code learning can push us into impostor syndrome, can make us feel we don't belong (particularly if we're not a 19-year-old white male in a hoodie), can make us feel frustrated and anxious and overwhelmed. You probably will feel that way if you learn to code, and that's okay. One of the biggest things, in fact, that learning to code will give you is a toolbox for handling those feelings and the knowledge that you can do the work even if you're intimidated.

"Very recently, a cataloging support staff member presented me with a printout of one of my old OCLC Macro Language cataloging scripts. The script produced a template MARC record for a title from a specific e-book collection, and she had edited it, largely correctly, to make the record it produced comply with new Resource Description and Access cataloging practice. She had 'discovered' programming by way of one of my scripts—this was very thrilling to me!"
—Carrie Preston

This, however, is a big reason that it's good to *find a mentor*. Mentors are great for answering technical questions and for telling you about tools and best practices that may not be written in books. But they're also great for holding your hand, cheering you up, and bolstering your self-confidence.

"You have to keep persisting. This is very different from writing a LibGuide or a handout."
—Bohyun Kim

Where do you find one? If you have a friendly, technically skilled colleague at work or a nearby institution, that's ideal. Some institutions (e.g., the George Washington University and the University of Maryland) have even started regular code-learning groups for their librarians. If you can't find a nearby colleague, the numerous technology-focused library conferences are great places to meet people. Nonlibrary technology can also be a good place to look. Many technical groups organize on Meetup.com; look for nearby meetups focused on your technology of choice. Be aware, though, that not all are beginner-friendly, and some can be downright hostile to women or people of color; look for groups that have outreach events, codes of conduct, or other clear commitments to hospitality. There are also technical groups focusing on outreach to specific populations that may be relevant to you, like PyLadies, PyStar, RailsBridge, and Trans*H4CK. All of these groups (plus ones focused on outreach to children, like Black Girls Code) are constantly looking for meeting space; if your library can offer some, that's a great way to build bridges to your local technical community, too.

Meetup.com
www.meetup.com

PyLadies
www.pyladies.com

PyStar
http://pystar.org

RailsBridge
www.railsbridge.org

*Trans*H4CK*
www.transhack.org

Black Girls Code
www.blackgirlscode.com

Finally, while in-person mentors are generally better, it's okay if you don't have access to them; the mailing lists and IRC channels for Code4Lib, LITA-L, LibTechWomen, and the like can expose you to current thinking and give you a place to ask questions. LibTechWomen has been running Code Club discussion groups; it's easy to set one up yourself by following Saron Yitbarek's advice.

Saron Yitbarek, "Reading Code Good"
http://bloggytoons.com/code-club

You may have noticed there's one question many beginners ask that I didn't answer here; to wit: "What language should I learn?" That's because there's no one answer to this question. Survey respondents wrote library code in fourteen different languages. The best language for you to learn depends on your personal taste, whether you have ready access to a community of experts, and above all the project you want to write. If you're modifying existing code or participating in an established open-source project, the choice of language is already made. If you're starting from scratch, your choice of project still influences your choice of language; for instance, web development probably means JavaScript, and MARC processing wants a language with an established MARC library, like ruby-marc, Python's pymarc, or PHP's File_MARC. Look for projects similar to the one that you want to do (including the projects in this report) and use their language choices as a guideline.

Finally, what tools should you use for learning? Google and Codecademy came up frequently in survey respondents' recommendations. While they have value (and Google is indispensable), as a teacher of code to librarians, I'm skeptical of unstructured and unsupported learning experiences. Because there's so little formal pipeline for teaching librarians to code, those librarians who do are, almost definitionally, the ones who do well with self-teaching, and their recommendations demonstrate a certain survivorship bias. I believe many librarians who aren't already coding, but want to, are more likely to succeed with a more structured, social experience. I've also been more generally impressed with the curricula in O'Reilly books than in free online courses; whatever your language of choice is, O'Reilly almost certainly publishes an introduction.

ruby-marc
https://github.com/ruby-marc/ruby-marc

Python's pymarc
https://github.com/edsu/pymarc

PHP's File_MARC
https://github.com/pear/File_MARC

Other specific resources recommended by respondents include:

- *The Art of UNIX Programming*, by Eric S. Raymond, https://openlibrary.org/works/OL6036022W/The_art_of_UNIX_programming. Many librarians find that command-line tools are even more useful than programming languages.
- *_why's (Poignant) Guide to Ruby*, a *sui generis*, part-cartoon introduction available free online, http://mislav.uniqpath.com/poignant-guide/book.
- *Python Programming in Context*, by Bradley N. Miller and David L. Ranum.
- The Pragmatic Studio, "Ruby Programming," online course, $132 with discounts and free trial available, http://pragmaticstudio.com/ruby.
- Lynda.com courses, www.lynda.com. In my experience, these are somewhat advanced for beginners, but excellent if you have a bit of prior experience, or good mentors; many libraries have a subscription.
- Google's Python Class, https://developers.google.com/edu/python. This resource is also best suited for people with some background; it is free, with good practice exercises.
- Formal courses available at your institution or in your area. These will probably be more theoretical than many librarians want and will likely not address library use cases, but taking even one will make it much easier to get mileage out of free resources.

It's also worth noting that several respondents said you should *not* try learning to code—or, at least,

"For the, the big thing was *find the right introduction*. There are a lot of guides for learning to code around, many of whom assume this or that reason why you might want to program, or start with the assumption that you have pre-existing knowledge of how to program. I learned to program from *_why's (Poignant) Guide to Ruby*, and I think this sentence is the very moment it clicked: 'You will be writing stories for a machine.' Coding as creative act, as artwork. Not algorithms or math or business rules. That caught my attention, and that got me going."
—Misty De Meo

that you should do it only if you're genuinely passionate about it and not just to check off a line on your resume. They indicated that people without this passion either would not succeed or would not become very good coders (and they felt that the world does not need more low-skilled coders). I agree in part and disagree in part. Coding is challenging enough that commitment is necessary; if you don't have that commitment, by all means spend your time on other things—there's no shortage of skills that will enrich your life and work! And becoming a deft, insightful coder is a full-time pursuit, and thus out of scope for most librarians. On the other hand, as we've seen in this report, you don't need to write large-scale, polished, reusable software in order to get big benefits from learning to code. Automating a task with a few dozen lines of code can save you many hours in a year. Even if you're a barely adequate coder, you can spend those extra hours being a fabulous original cataloger or research consultant or department head, employing human judgment and doing tasks the computer can't.

Workplace Support

Because learning to code can be time-consuming—and because librarians' code skills can be so beneficial to their institutions—it is both helpful and relevant for librarians to receive professional development support in learning to code. I asked respondents what, if any, workplace support they had received; I also asked managers what, if any, they had provided or would provide.

Answers varied significantly. While managers who code understand uniformly the value in supporting this skill, not everyone is lucky enough to have such a manager. Among institutions that do support code learning, funding and policy vary. Among survey respondents, the gold standard was set by Evviva Weinraub Lajoie at Oregon State University Libraries & Press. She provides employees with twenty hours per month of learning time, at least one conference per year, access to paid online tutorials, and even structured internships. Other libraries can't offer this level of support, but at least provide informal mentorships, code review, and the like.

Unfortunately, some librarians have no support, or even face active hostility. Some institutions simply don't have pertinent checkboxes on their paperwork, and it's hard to pertinent the relevance of these skills to a faceless bureaucracy. Two managers were unable to secure coding skills development for interested supervisees because their institutions did not want to reclassify them into higher salary categories reflecting those skills. And, as we saw in chapter 5, one librarian who spends a significant amount of time coding is doing so without upper management's knowledge; in that institution, only people belonging to other, explicitly technical, units are allowed to code. (Middle management "works pretty hard to keep me writing code as much as possible, even letting me out of some regular meetings because they know I can contribute more if I'm tickling a keyboard," says this librarian, who will remain anonymous for obvious, though distressing, reasons.)

Tech-savvy managers uniformly recognize the value of these skills and are willing to support them. Not all of them have supervisees who are interested, and the availability of funds varies, so the specific support provided does also. However, types of support that managers provide include:

- time: finding ways for planned projects to include learning new technologies, setting aside time for learning and experimentation, defending this time to upper management
- books
- software licenses
- root privileges, development sandboxes, testing servers, quality hardware: in short, the ability to install and experiment with software
- conference attendance: supported in time, money, or both
- workshops: some paying for attendance, others teaching them personally
- regular study groups, such as the one at the University of Maryland libraries or the George Washington University code reading group
- courses: online (such as Lynda.com, Code School, RailsCasts, Treehouse) or face-to-face, through tuition remission in the case of academic libraries
- code review
- mentorship
- formal internship programs
- making coding skills part of supervisees' performance goals, which helps justify other forms of support

University of Maryland Libraries Coding Workshop
https://github.com/umd-coding-workshop/website/wiki

Lynda.com
www.lynda.com

Code School
https://www.codeschool.com

RailsCasts
http://railscasts.com

Treehouse
http://teamtreehouse.com

Conclusion

Throughout this report, you've seen how librarians use short programs to make their work lives better in concrete ways, the opportunities (and obstacles) posed by code, and strategies you can use to start learning or to upgrade your skills.

Now it's your turn! Pick a project, find a class, put together a study group: whatever your next steps are, get started.

Whatever you do, you can always find source code for the projects discussed in this report, plus others that didn't fit—including the source code for the Django app I wrote to keep track of my own survey data—on the companion website. If you have a project you'd like to share—particularly one you wrote as a result of reading this report!—I'd love to feature it there as well; instructions are on the site.

Companion website
https://thatandromeda.github.io/ltr

Notes

1. Kate Ray, "Don't Believe Anyone Who Tells You Learning to Code Is Easy," TechCrunch, May 24, 2014, http://techcrunch.com/2014/05/24/dont-believe-anyone-who-tells-you-learning-to-code-is-easy.
2. Cecily Carver, "Things I Wish Someone Had Told Me When I Was Learning How to Code: And What I've Learned from Teaching Others," Medium, November 22, 2013, https://medium.com/@cecilycarver/things-i-wish-someone-had-told-me-when-i-was-learning-how-to-code-565fc9dcb329.