Curriculum for Learning to Program

Oct. 5, 2011

http://evbeard.com/posts/learning-to-program/

Introduction

Many people have asked me to teach them how to program, so I have created this mini-curriculum. Learning to program, and becoming a good programmer, is a largely individual endeavor that requires much self-study. But the trouble with getting started is often that you don't know where to start or what to topics to learn. This document aims to solve the problem of not knowing where to start by providing a suggested itinerary of topics valuable for someone who wants to learn practical programming skills.

This document is intended for those who are interested in starting a technology company but have little or no technical background. Even if you never intend to write code for your startup it is vitally important that you learn to program yourself. It will allow you to know what's possible when determining your feature sets. It will arm you with discernment when comparing engineering candidates, for having strong engineers is vitally important. It will help you to manage your employees.

It is not too late to start. You can learn to program and become technical, it's only a matter of determination. You should start today. having said this, programming is not for everyone. The main requirement seems an ability to concentrate for a long period of time. Some personality types aren't well tuned for this and may not make for good programmers. But this doesn't prevent you from following along to learn the basics.

This is a work in progress. It is the curriculum of topics and skills that I put together to teach my brother how to program. He found it useful, so I'm sharing it in the hope that others will as well. It's intended for people looking for a thorough curriculum and have time to invest in learning. The topics I recommend are of course my own opinion. Please send feedback to ev [at] nbeard [dot] com.

How to use this document

You will learn the topics herein by Googling. **You will need to Google everything.** You will need to google for tutorials, walk-throughs, specific error messages, etc. for the topics listed in this curriculum. Please see here.

Learning to program is like learning a foreign language: you first study the material, then learn by trying to use the knowledge that you just learned, then repeating. So it is critically important that you write code from the very beginning as you are learning. And don't forget to read code; you'll learn a lot by reading other people's code.

Although not required, it would be very helpful for you to have a technical friend on-call to answer questions when you are very stuck. Listed here are the topics that you should learn, but you will inevitably come to an impasse when doing something. If you can email someone technical they may be able to fix a problem with only a few seconds of thought that could otherwise take a beginner hours to solve. A second way a friend can help is to setup post-commit emails on github so they are emailed whenever you push code and know to visit github to perform a code review for you. Having your code reviewed by an experienced programmer is a great way to improve quickly.

Preparation

Choosing mac (or linux) vs windows

Ideally you'll be on OS X or linux because they're based on unix, which is what the servers your code will be deployed to will run, so it's best if your local environment is the same as production. If you only have access to a windows computer, you can simulate a unix environment on windows with cygwin (or you can dual-boot Ubuntu).

Accounts to create

- Github - a way to collaborate with other people on the code you are writing, and keep your code "versioned" so you can revert back to old versions and save your work regularly
- Amazon EC2 - this is a way for you start servers that can do things such as host a website, or be the backend for a mobile application

Programs to Download

Mac or Windows:

- Firefox and Firebug (firebug is a tool for web development that makes it easy to see what visual elements on the page correspond to what HTML/CSS, to quickly preview HTML/CSS changes, and to trace javascript errors and execution)
- Chrome (has a developer toolbar built in similar to firebug)
- Photoshop/Illustrator (start with 30 day trials for both) (programs to create images)
- cyberduck (for SFTP -- a way to transfer files to your EC2 servers)

Mac:

- Sequel Pro (a GUI to MYSQL)
- git (a client for version control)
- homebrew (a package manager, aka an easy way to install programs)
- OS X developer tools / xcode (an IDE for OS X / iphone development that has the side effect of also installing a C compiler that you will need to compile / install some programs)
- colloquy (an irc/chat client for getting help from other programmers)
- Add the Terminal application to your dock for easy access (go to spotlight in the top right and searh for "terminal" to open it, then right click and go to "options"->"keep in dock")

Windows:

- cygwin (simulates a unix environment)
- in cygwin: install the git-core package (type "git" in the search box) and install ssh
- komodo edit (the free version) (a text editor / IDE aka a program to help you code)
- mIRC (an irc/chat client for getting help from other programmers)
- putty (an SSH client -- allows you to remotely log in to your EC2 servers so that you can run your code once it is transferred to the servers via e.g. cyberduck)

Background Task: Watch intro CS courses

Watch these courses while simultaneously going through the other tasks.

MIT Introduction to Programming

Stanford Introduction to CS Courses

Lesson 0: Desk setup

Why do this? Without a comfortable setup you'll have trouble concentrating and could experience back, wrist, or forearm pain.

Ergonomic evaluations generally aren't that useful - but one ergo tip to follow is to keep your arms at right angles with the table.Your forearms should not touch the table. If your forearms have contact with the table at the wrong angle you can begin to experience pain or RSI. See this photo. I never realized how important this was until I actually had a comfortable setup.

You may consider a standing desk to help break up long periods at your desk. And I recommend taking breaks to walk around for about 5 mins every hr or two unless in the middle of something, 15 mins every few hrs.

Lesson 1: Learning Unix

Why learn this? When you are developing you will need need to use the "command line" (which is the Terminal app on OS X). You'll need to know how to run a server on your development box and interact with it. When it comes time to deploy your application to the world, you'll need to use the same command-line interface to connect to a remote server and set up your service to be publically accessible.

Tasks

- Google for and follow unix tutorials. Make sure you understand the idea that you are always in a specific folder when you are at a command prompt (just as you are when you have an open finder window on os x).
- Open a terminal and learn the "man", "ls", "pwd", and "cd" commands. (Type "man [command]" for help with a command).
- Learn how to use the "SSH" command (search for "ssh tutorials" and look at "man ssh").

Resources

- this 10min YouTube video covers the basics

Lesson 2: Learning HTML/CSS/Photoshop/Illustrator/Git

Why learn this? HTML and CSS will allow you to create websites. Photoshop and Illustrator help you to create graphics, whether for a web, mobile, or desktop app. Git is a version control system that will allow you to revert to previous versions of your work and collaborate with others.

Tasks

- create a simple HTML / CSS website that looks as close to the google homepage as possible (except you should create a different logo with a different name in photoshop/illustrator).
- Bonus points: make it so that when you click "search" it loads the google search results for whatever you type in the box (hint: you need to use a form - action type GET)
- Use photoshop and illustrator to create your own custom logo for your search engine (to replace the Google logo)
- Push your project to a github repository (instructions below for pushing all changes)

- Learn how to deal with conflicts in Git.

to push all of your changes to github (though with experience you won't push all changes so blindly):
> $ git add . #(to add new files)
> $ git commit -am "Here is the message”
> $ git push origin master

to pull changes from github (if you have another programmer contributing):
> $ git pull origin master

Resources

- http://book.git-scm.com/
- http://sixrevisions.com/resources/git-tutorials-beginners/

Lesson 3: Learn Emacs

Why learn this? Emacs is a very powerful text editor. Though there is a steep learning curve to know all of its capabilities, I find that one can learn the basics quickly and benefit from the power of emacs. Finally, emacs is a command line editor, so when you are remotely logged in to your server you will be able to use it to edit files that are on your server.

Tasks

- install emacs. Aquamacs on OS X is a good way to learn because it allows you to use the mouse while learning (eventually you won't use the mouse at all because it's slow)
- go through emacs tutorials, learn hotkeys for emacs, learn how to open files with it without using the mouse, learn how to create multiple buffers.
- learn how to use contrl-x f to open files
- learn how to undo/redo/about the undo ring
- learn how copy and paste works (it's different than you may be used to)
- learn how to navigate around a document (you navigate not by using the mouse, but by scrolling up/down (see commands below) or by looking at the place you want the cursor to be and then doing a search to relocate the cursor to where you want to go)
- learn how to use the following hotkeys:

control-a (go to beginning of line)
control-e (go to end of line)
control-k (delete line)
control-y (paste line)
control-v (scroll down) meta-v (scroll up)
control-s (search/navigate)
control-r (reverse search/navigate)

Resources

http://www2.lib.uchicago.edu/keith/tcl-course/emacs-tutorial.html

Lesson 4: Learning Server Admin

Why learn this? Whether you are building a web site or a mobile app, chances are you will need to set up a publically accessible server. .

Tasks

- Create an EC2 micro spot instance running ubuntu (a linux distribution) using amazon's ec2 web console
- Download the key, and remember where you save it
- https://help.ubuntu.com/community/EC2StartersGuide
- Using the command prompt, navigate to the folder in which you saved your key then you log in with something like “ssh -i keyname.pem ubuntu@here_is_the_public_dns”
- Once logged in type “sudo” to become root. Then use “passwd” to change the root password so you can log in with a password from now on.
- Figure out how to log in to your instance, create your own user and password.
- Learn how to use ubuntu’s package manager (apt-get) to install apache (for help google: “how to install apache on ubuntu”).
- Transfer your files over using SFTP (either through command line or cyberduck).
- Put your files in the web root (usually something like /var/www, but use a tutorial).
- Make sure you can see them at the public dns of the micro instance (which you can find by clicking on your server in amazon's ec2 web console)

Lesson 5: Learn DNS

Why learn this? So that you know how to associate domain names with the servers that you instantiate.

Tasks

- Think of a name for your project by using whois to find available names (I like http://ajaxwhois.com/)
- Create a dreamhost account and purchase name
- In dreamhost: create an A record that points from your domain name to your EC2 server IP address
- Make sure you can view your website at the domain name that you purchased

Lesson 6: Learn Python (your first programming language)

Why learn this? This is a good first programming language, both for its approachability to a beginner and for its ability to scale with you and run your first site in production.

Tasks

- Write a CL (command line) program that tells me how long it will take me to lose weight. You should write the program in Aquamacs/Emacs or a similar program. Save the file as .py in order to change the application into python mode. It should ask me my base metabolic rate (the number of calories i burn per day on average without exercise), the amount of calories i eat per week, the number of workouts i do per week (assume i burn 200 calories per workout), and the number of pounds of fat i'm trying to burn. assume it takes burning 3500 calories to lose one pound of fat. Your should print the length of time in days, weeks or months, whichever is appropriate eg. "1 week" or "2 days" or "1 month". Type "python filename.py" into the terminal in order to test the program.

Resources

- http://wiki.python.org/moin/BeginnersGuide/NonProgrammers
- http://docs.python.org/tutorial/
- learning python book

Lesson 7: Learn SQL (your first declarative language!)

Why learn this? SQL is the language of relational databases, and almost every webapp you use (and most mobile apps) have a relational database behind them. By knowing how to interact with a relational database, and design a database schema, you'll be one step further to making your own apps

Tasks

- mysql is the most popular free/open source database. facebook uses it, as well as most startups.
- install mysql on your mac and start going through mysql tutorials
- tip: use sequelpro to interact with your local mysql database
- load this dump file into your database: http://www.cactusflower.org/2009-salary-data This is 2009 salary data from the government.
- Tell me the states with the highest employment, the highest salary.
- which jobs pay the most?
- Learn how to do joins, and tell me some things you learn from doing joins.
- read about database design so you know how to make your own tables
- read about denormalization and understand when it's appropriate

Resources

- google for mysql tutorials

Lesson 8: Learn Django (a web framework)

Why learn this? If you were to build a dozens websites, many of them would share the same code: a system for users to create accounts, to map urls to function calls, to interact with the database, etc. Web frameworks such as django provide these common tasks so that you don't have to write extra code. When you are first learning

they are especially good for pointing you towards best practices.

Tasks

- setup a django app that allows me to sign in with a username and password, and then lets me keep a list of things to do. i want to be able to add to my todo list and delete todo items from the list. use mysql as your backend database (so you will need to now set this up on your server).
- put your django app on the web so i can use it at yourdomainname.com.
- how to debug with python, how to run a python program, note for django to use emacs, print or pdb. for django lesson -- just run django locally at first. add how to deploy in the doc. use django book
- Read the code of django admin app and https://www.django-cms.org/
- join the #django channel on freenode IRC and ask a question when you're stuck on something (use colloguy for mac)

Resources

- https://docs.djangoproject.com
- http://www.djangobook.com/en/2.0/
- https://docs.djangoproject.com/en/dev/intro/tutorial01/

Lesson 9: Learn javascript

Why learn this? This is the language that runs in web browsers. If you have ever used a highly interactive web application, such as gmail, you've experience what javascript can do in the browser. You can create animations, load data remotely, and do all kinds of other things.

Tasks

- as always, start by reading javascript tutorials.
- start reading jquery tutorials, you will be using jquery.
- now make your django todo app work over ajax using javascript / jquery.

Resources

- http://www.amazon.com/JavaScript-Good-Parts-Douglas-Crockford/dp/0596517742/ref=sr_1_1?ie=UTF8&qid=1316727962&sr=8-1
- https://developer.mozilla.org/en/a_re-introduction_to_javascript
  https://developer.mozilla.org/en/JavaScript/Guide

Lesson 10: Learn basic algorithms / algorithmic complexity

Why learn this? This is important so that you know how to write code that runs as quickly as possible. If you write inefficient code your application could be slow or may not run. It's also important so that you know when a problem is one that may be impossible to solve in a reasonable amount of time.

Tasks

- Watch MIT Introduction to Algorithms
- Understand computational complexity (http://en.wikipedia.org/wiki/Computational_complexity_theory)
- Learn Big-O notation (http://en.wikipedia.org/wiki/Big_O_notation)
- Understand minimally what a hashmap is, what it is called in different languages, and why it has constant time lookup. Understand when to use this constant time lookup quality of hashmaps to save time (e.g. instead of looping over a list to find an item).

Resources

- MIT Introduction to Algorithms

Lesson 11: Learn refactoring

Why learn this? You must write clean code. You can get away with ugly code when you are the only one working on your particular section of code and the codebase is small, but as your codebase grows or you add more collaborators development will slow to a crawl if you are not writing clean code and constantly refactoring.

Tasks

- Google refactoring and read all about it

Resources

- Refactoring book

Lesson 12: Learn some Java (a different type of programming language) and object oriented programming

Why learn this? Higher level languages such as python aren't the right tool for every job. You should know at least one strongly typed language such as python.

Tasks

- read about object oriented programming
- read about java, check out java tutorials and write something using java

Lesson 13: Learn how an open source project is managed

Why learn this? Learning how an open source project works will make you aware of many parts of the software development process that you may not have yet considered, such as tracking bugs.

Tasks

- subscribe to the django developer mailing lists (stay on it for at least 1 week), look at the django bug tracker, read the docs on how to contribute to django

- find an open source project (it can be anything that interests you) on github and submit a patch (to either fix a bug or write a new feature).

Lesson 14: Learn how to plan for a project

Why learn this? The right planning can save you a lot of time in the long run.

Tasks

- creating mockups, planning timeframes, creating the minimal viable product.
- Before writing code for any serious project you should plan out the project out very thoroughly...know what APIs you will use, know what libraries you plan on using, what techniques/algorithms you will use for the hard or not obvious things the product will do. And then you should mock everything out (using a tool like balsamic or tiggr) so that you know exactly what you're building. This will save a lot of time later from throwing away code because something isn't possible, or because you need to restructure the UI (user interface) due to a different technique you need to use.
- An architecture review / design doc will allow you to get feedback from more technical friends, which can save you much time by preventing you from rewriting in the future
- Usually we plan and build the minimal and then add more features in layers so that we can quickly get something up that works. Once you've written your design doc revisit your mockups to make sure everything in them is possible and that you didn't overlook items that require changing parts of the UI.
- Get feedback! Feedback from your circle of trusted friends / family is extremely valuable, even if they aren't your target audience. Specially your non-technical family members and friends are valuable for feedback on mockups. A second pair of eyes can identify confusing parts that seem obvious to you, or show you a much easier way of doing things and save you a lot of time. In larger companies there are entire teams to help with recruit and facilitate getting feedback on mockups.

Highly Recommended Reading

- Code Complete 2
- Design Patterns

Daily Reading

These sites will keep you in-touch with current tech trends, new technologies and libraries, and will serve as a source of ideas for projects to work on

- Hacker News
- Techmeme

Other topics

These are topics that you may want to explore as you learn more

- writing unit and functional tests
- test driven development
- threading

- functional programming
- key-value stores (e.g. HBase)
- message queues (e.g. RabbitMQ)
- mobile development (e.g. android/ios)
- serialization formats (e.g. XML/JSON/protocol buffers)
- event-driven programming
- scrum / extreme programming

Here are some modern software projects. It could be beneficial to look them up, understand when you would use them, and perhaps even read through their source

- mongodb
- cassandra
- hadoop
- redis
- memcached
- twisted python
- ruby on rails
- nginx
- subversion
- PostgreSQL
- RabbitMQ
- Node.js

Here are useful websites

- http://highscalability.com

Conclusion

You will not learn to program overnight. It will take thousands of hours to really know what you're doing. The trick is to really enjoy yourself so that you want to do it, and to do it every day. Do not get discouraged, and do not give up.

You'll need a project to work on to help you continue to learn. It can be anything, as long as it's a business or project you're passionate about. Even if it doesn't work out you can use your newfound technical knowledge for the next project.