Some ideas about computer programming, for the Dickinson College Idea Fund

John MacCormick, April 13, 2015

The intended audience of this discussion is people who are interested in social innovation and/or entrepreneurship, are planning to pursue a project in this space that incorporates some technology or software aspects, but who have little or no background in computer programming or software development. The main objective of our discussion is to provide a few pointers and some high-level advice on initiating and managing such projects, and how to build up a useful minimal core of knowledge in the area of software engineering.

Here are some of the specific questions I was asked to address:

1. What are the different coding languages? What are the differences between them? What are their different applications?

Interesting fact: all programming languages are equivalent in the sense that they can, in principle, all solve the same set of problems. (Yes, this is a mathematical theorem that has been proved -- see the Wikipedia page on "Structured program theorem"!) So, why do we care about differences between languages? Because some languages allow you to *write* (or *develop*) a solution to certain types of problem more quickly. And some languages will *run* (or *execute*) the solution (after you have finished writing it) more quickly than others. And some languages work particularly well for niche applications. A very rough categorization of some popular languages would be:

C++ -- a little challenging for development, very fast for execution. General-purpose, also particularly good for low-level applications like operating systems, embedded systems etc.
Java -- easy for development, a little slower for execution. General-purpose. The only choice for developing Android apps.
Objective-C -- similar to Java, but the only choice for developing iOS apps.
C# -- similar to Java, but developed by Microsoft and lies the Microsoft ecosystem. Use for Windows 8 apps, for example, but also any Windows desktop application.
SQL -- specifically for databases. It is used only to execute queries on databases, not for general-purpose programming. Typically used on the back end of a website, providing information to the front end.
PHP -- easy to develop, a little slow on execution. Particularly popular for Web servers to generate web content (the "front end").
Javascript -- completely separate from Java, despite the similar name. Well-suited for running a program *inside* a client's web browser (i.e. not on the server – on the client's own device). However, it is a general-purpose language and is used for server side also.
Python -- very easy to learn and develop in, but a little slow for execution. Popular for "gluing" parts of a system together, but is also used as a major development language.

[C++ and Java are the most attractive languages for employers in general. However, employers seeking web developers will be looking more for people who know PHP and SQL.]

2. If you are someone who isn't familiar with coding but wants to be, where can you go to get resources to help you with coding?

My answer here is a little conservative and arguably negative, but I think it’s important to acknowledge the facts. Achieving genuine fluency in a computer programming language is easier, for most people, then achieving fluency in a human language (such as French or Japanese) -- but not by much. It is extremely challenging to become a skillful programmer, requiring substantial effort over a long period of time. A realistic minimum would be 2-3 college courses to achieve a level of basic competence. Of course, something is always better than nothing, and there is now an immense amount of free high-quality online resources available for learning programming. There are so many good ones that it’s unfair to single anything out, but if you want concrete recommendations, I would suggest a few possibilities:

- The CS50 course from EdX -- this is, more or less, Harvard’s introductory programming course, and it covers a range of different languages and ideas. But you’ll need to invest the same amount of time in this as in any other college course.
- Download Python from python.org, and start working through one of their online tutorials. Or buy a Python book and work through that. Python is probably the easiest language to pick up and get some idea about computer programming. This is the best option if you are only willing to spend a few days or weeks gaining familiarity.
- Popular online options include Codecademy, Khan Academy, and the main MOOC companies (Coursera, Udacity, and EdX). I have not investigated most of these in detail, but a brief perusal suggests they all provide high-quality opportunities for learning to program and acquiring computer science concepts. As above, I recommend Python as the easiest beginner language, so it may be best to select a course that uses Python as the main teaching language.

3. If you are someone who isn't familiar with coding but wants to be: what are some basic concepts you should understand in order to talk to someone who might do some coding for you?

The productivity difference between a skilled programmer and a mediocre programmer is really huge -- far greater than the difference between skilled and mediocre practitioners in many other fields. (This is very well documented – do a web search on “Productivity Variations among Software Developers”.) Therefore, the most important thing to ascertain is whether your programmer is a skilled programmer. This can be achieved by the same methods as assessing the skills of any other potential employee or partner: namely, checking references and conducting a detailed interview. You don’t need technical knowledge to ask questions such as, “We hope you will work on project X. What’s the most similar project you’ve worked on in the past? What was the outcome of the project? What’s something you would do differently if doing it again?” If you are planning to work with a student who has relatively little experience, then you have to accept a reasonably high level of uncertainty on whether or not the outcome will be successful. Seeking recommendations from other students and/or instructors could help.

4. What are some everyday applications of coding that a non-computer science person may run into?

You can’t avoid running into it. To mention just a few of my favorites that you use every day: cryptography for any secure interaction over the Internet, error-correction codes for transmitting error-free information over a noisy wireless channel, and sophisticated ranking algorithms used by web search engines.