Computer Science Literacy

Rushi Shah

7 November 2016

One of the courses every student at my high school was required to take was "Foundations of Computer Science". I ended up taking for granted that my peers had a certain level of computer science literacy that was surprisingly useful in explaining topics that seem orthogonal to computers in general. Now that I attend a school where no such requirement is imposed, I've noticed how useful general computer science literacy would be.

With that being said, I don't generally consider it valid that in the coming years everyone will need to know how to program their own solutions to problems. That would certainly be nice, but I think as of right now it seems a little unnecessary.

Instead of expecting everybody to be programmers, I expect everyone to have a base level of understanding of how the Computer Science field works and can be useful to their lives past the finished products they use. I equate this idea to how everyone knows a bit of math. Arithmetic corresponds to knowing how to use a mouse, type, and just in general get around computers. But a lot of people know a lot more math than just basic arithmetic. Here is an outline of the topics I think the general populus should know and a bit of why:

1 Types

I can't emphasize this enough, types are the core of Computer Science! I genuinely believe that type intuition is the most applicable aspect of computer science to the non-tech community, which is probably a pretty unpopular opinion. I argue that every domain has an implicit type system. And recognizing this type system can clarify why some things are wrong that we have accepted are wrong.

For example, in my introductory physics class, my teacher always emphasized that you should check that the units on your final answer correspond to the units the question is asking you for. This dimensional analysis makes sure you don't give speed (distance per time) when the problem asks for just distance. Another related tip is to make sure that every piece of provided information has been "used up". In the example, this would imply that there is probably some time variable you forgot to take into account and multiply by. This is just a type system!

Similarly, in grammar, some sentences just don't make sense. If you replace a noun with a verb, everything is just going to sound wrong. Reasoning about the types of speech can help you learn a new language, etc. These parts of speech represent yet another type system!

The literacy I propose doesn't need to be perpetuated because of these examples (physicists and linguists don't necessarily need to understand the type system to understand what

they're doing). But these examples demonstrate a larger point I'm trying to make: types are everywhere. Educating people about types in a more concrete sense will let them identify more abstract representations of types in their life.

What do I mean by educating people about concrete type systems? I think every high school graduate should at least understand what a boolean and string are. They aren't difficult concepts to grasp once they've been explained, but without the requisite knowledge those two words could seem intimidating. Its not a matter of destroying the jargon (which in this case I argue there is little of), but rather of introducing concepts. To return to the math literacy example: people generally know what "probability" is, but if they didn't, it would sound like a big scary word.

And the cool thing about types is you can demonstrate them in a REPL, the person doesn't even have to write programs to see why you can subtract two integers, but not necessarily subtract two boolean values. So I propose people learn about the basic types: booleans, numeric types like integers/floats/doubles/etc., and strings. Then, maybe learn a smattering of functions to combine these basic types together and show why types can help the computer reason about programs.

2 Basic Data Structures

Okay great, people know what a String is. So like you can store your name in a variable. But what if you want to store everybody in your class's name? Would you need a variable for each person? Of course not, there has to be a better way to store lists of names. So everybody knows data is a thing, but perhaps they don't consider the fact that you can take a bunch of data and structure it. For example, you can take a list of names, and store them as (surprise, surprise) a *list* of Strings.

I would propose that the next step is at least teaching people what lists (arrays) are as an introduction to the concept of `data structures`. If I wanted to be optimistic, I would also recommend a few other data structures, namely trees and graphs. Trees represent hierarchacal data that is overwhelmingly common and unique to other types of data structures in the way that you can consider subtrees at a time. Graphs will give the notion of having pieces of data point to other pieces of data, which is a valuable intution.

The emphasis here is not implementations of data structures, efficiency, or any such nonesense. Instead, we are trying to talk about data structures in a more abstract sense. We want to demonstrate why we need them and how they are useful.

3 Functions

What are computers good at that humans aren't? Doing the same repetitive task over and over (perhaps with slight tweaks) quickly, efficiently, and accurately. This is where functions come in, and they are FUNDAMENTAL to computer science.

4 Basic Control Flow

At this point, hopefully computer science will seem a little less daunting. Now that they know the atoms of programs, the student should start messing around with it a little bit to see that "hey this isn't so bad!". In the process of doing so, teach them if-statements to use those booleans they learned about. Teach them about iteration to use those lists they learned about. And maybe if you included a section about functions, include an addendum about recursion to use the functions.

4.1 Iteration

"Keep doing it until". For each element in a list.

4.2 If Statements

Booleans can represent our world. From simple (is it past 5 o'clock) to complex (is a user logged in).

4.3 Recursion

Conversations are recursive. When you finish doing something, start over.

5 Conclusion

I haven't hammered out the correct order yet, because should students learn functions first or data structures first? And should they start programming early or later? And should we integrate the iteration in more tightly with the lists and the trees with the recursion? There are just a lot of unanswered questions, and I think the biggest source of confusion for me is when should things be taught and what things should be taught together.

Regardless, these are the bases that should be covered for so called "computer-science literacy". Just having a general sense about the words and what they mean can help the lay-person get their bearings in the Computer Science field and figure out if it is something they want to pursue further.