What is programming?

Attila Egri-Nagy

v2017.12.30

Instructing computers? Writing text using arcane symbols? Pastime activity for the socially deprived? Money making career choice? Does it bring doomsday closer or is it a key skill for our survival? Here is an idiosyncratic list of a few topics towards answering these questions.

On the surface, programming is only about instructing computers to perform useful tasks. In other words, programming is creating and controlling computational processes. We can often break down a big task into small steps, where these operations are so simple that a physical process can carry them out. Programming is a clever way of combining these simpler steps into more useful ones repeatedly. As a result, we can make the computer carry out complex tasks that are menaningful to us. This is already very important, since we have surrounded ourselves by computers. But a closer look reveals that it is also a cultural phenomenon. It has a rich history, different ways of thinking, its own aesthetics, and programming even makes sense without any computers around.

Philosophy and Programming

During my undergraduate years I studied Mathematics, Software Engineering and Philosophy in parallel. Many people asked about this combination. How is it possible to mix extremes, such as ancient Greek metaphysics and programming? My view might have been a bit peculiar, but I failed to see the difference. Our goal is to understand the world around us by creating an abstract model of it. The way of Philosophy is to discuss as many wild ideas as we can conceive, and see what makes sense or simply sounds good. The programming approach is to explain to a computer how the world (or some part of it) works. Explaining helps a lot in understanding someone's own ideas. Since instructing a computer requires very high precision, programming forces us to understand the phenomenon we are trying to model computationally. So, the philosophical method and programming are different, but the purpose is the same.

Declarative vs. procedural knowledge

One thing is to know what $\sqrt{2}$ is. It is the number that gives 2 when multiplied by itself. It is a different thing to know how to obtain its actual value. Here's a method: start with a guess, let's say 1. Then

$2/1 = 2$ shows that our guess is not right, but we can improve it. By taking the average we get $\frac{2+1}{2} = 1.5$, a better guess. Now $2/1.5 = 1.3333$ and $\frac{1.5+1.3333}{2} = 1.4167$, getting close. By repeating this process we can get arbitrary close to the numerical value of $\sqrt{2}$.

Similarly, it is one thing to know how the solution of a sudoku puzzle looks like, and another one to know how to solve it. I know what I want, but I may not know how to get it. This is the distinction between the declarative and procedural knowledge. Some programming languages are more declarative, we only need to describe the solution we want. Thus understanding programming as instructing computers what to do is only a limited view.

Interestingly, while the underlying idea of programming is knowing how to do something, the development of programming languages is about gradually moving towards the declarative ideal.

Patterns of doing things

We can spot general patterns in our repetitive daily activities, and those appear in programming as well. Often we need to do the same action to a bunch of things.

For instance, we have a bag of oranges, and we peel all of them. In programming we say we *transform a collection*. Or, we might want to get all apples from a bowl of fruits. We go through the fruits and if it is apple, then we take it out, if not, we leave it in the bowl. This is *filtering a collection*. Staying with the fruity example. We can turn a bag of fruits into a fruit salad. We produce a single result from a bunch of things. This is *reducing/folding a collection*.

History, culture

Programming languages form families, and they are centered around paradigms. Imperative, object-oriented, functional, logic, to name a few major ones. People tend to gather around programming languages in communities to share ideas with like minded people and help each other. Sometimes these communities become tribal and aggressive to outsiders and members of other communities. This often manifests itself in discussing unimportant details, like what symbols to use to structure the code (e.g. `{}` versus `()`). It is more sensible to ask which language/paradigm is the most suitable for a given task, and just be nice to each other.

Text

Programming is about writing text. This can be viewed as a disadvantage or as a real deal. One can argue that we haven't made the breakthrough yet. Despite some effort for obtaining visual representations of processes, most programs are written as text. On the other hand, text can be viewed as the most precise way of storing information, especially how-to-do descriptions.

Typography

If it is text, then all the issues of typography applies to source code as well. The assumption is that reading requires mental effort, and the brain has to spend some of the energy for just recognising the letters and words. By making the typeface more readable, we can lighten the load.

Source code is written in monospaced (fixed-width) font, where each letter occupies the same horizontal space. This comes from the technological heritage of typewriters and it also has a practical purpose. The program is written in a grid, so it is easy to refer to exact locations by giving the row number and the character position.

Meaning

When we read a book, we imagine a world described in the book. Reading source code is similar. We imagine the computation which happens when the program is running. Sort of, our head acts as a computer. This is also where the difficulty of programming comes in. The cognitive load of understanding programs can be high, since computers don't have the same limitations we have. They do more things than what we can think of at a moment. Programming languages can be viewed as a collection of ideas to reduce this cognitive load.

To turn the metaphor around, writing literature can be viewed as programming minds (not in the political sense) to imagine events and characters.

Just like in natural languages, the meaning is often language independent. In programming we have the notion of the algorithm, then actual implementations in different programming languages.

Names

Naming is very important in programming. Not for the computer though, where names are translated into memory addresses, that

are just numbers. We could build some false mythology around programming as a creating act, pointing to religions and sorcerers, where a word can create worlds and conjure spirits. This might be entertaining, but the real importance is elsewhere.

Naming is a tool of abstraction. For a given complex structure or process, we just put a symbolic label on it, so we do not have to deal with its inner workings. Imagine a situation in which we are not allowed to use names for things, only to describe them – this happens when you learn a foreign language.

It is not surprising that programmers find naming difficult: you write a useful piece of code, that may be used often and by others, so the name should be a good summary, or a hint, or an indicator of what the code is doing. So the skill of naming is on par with being able to understand and communicate what a program is doing.

Understanding and Communication

If written code is never read by someone, then it is probably not used at all. No one cares to check it, no one is interested in that solution of the problem. In the learning process of programming it might happen that only the author of the program reads the source code.

In practice it is a form of communication. Team members can say to each other "This is how I think about the problem domain.", "This way I would like to solve this problem." through their written code. Writing source code is a form of communication.

Difficulty and the Thrill

It comes from the abstract nature of the 'material' software engineering is dealing with. Evolutionary, we are better equipped for dealing with, or even just imagining real objects. Abstract concepts, number for instance, need some spatial grounding (number line).

On the other hand, in the realm of computing, there are no physical limitations. We can build a little 'engine', a piece of code that works on a small piece of data, then we can immediately feed a data item that is million times bigger. In our physical world, constructions don't scale without limits.

Or rather, the physical limitations are not inherent, thay are due to the fact the computers are physical objects. Consider power consumption, cooling reuirements.

Importance

After several decades of living with computing technologies it is clear that they can be used both for benevolent and for malicious purposes, just as with any other advanced technologies.

Programming, as a way of understanding the world around us in terms of procedural knowledge, is crucial even without having computers. Learning to code is a way of improving thinking skills.