The Education Column

BY

Juraj Hromkovič

Department of Computer Science
ETH Zürich
Universitätstrasse 6, 8092 Zürich, Switzerland
juraj.hromkovic@inf.ethz.ch

Learn to Program? Program to Learn!

Matthias Hauswirth
Università della Svizzera italiana
matthias.hauswirth@usi.ch

Abstract

Learning to program may make students more employable, and it may make them better thinkers. However, the most important reason for learning to program may well be that it enables an entirely new way of learning.[1]

1 Why Everyone Should Learn to Program

We are in a gold rush in computer science education. Countless school districts, states, countries, non-profits, and startups *rush* to offer computer science, or coding, for all. The goal—or *gold*?—too often is seen in empowering students to get great future-proof jobs.

This first goal—*programming to earn*—is fine, but it is much too limited.

A broader goal looks at computer science education as general education that helps students to become critical thinkers. Like the headmaster of my school, who recommended I study Latin because it would make me a better thinker. It probably did. And so did studying computer science.

This second goal—*programming to think*—is great. However, I claim that there is a third, even greater, goal for teaching computer science to each and every person on the planet. Read on!

2 Computer Language as a Medium

In "Computer Science: Reflections on the Field, Reflections from the Field" [6], Gerald Jay Sussman (MIT) writes an essay called "The Legacy of Computer Science." There he cites from his own landmark programming textbook "Structure and Interpretation of Computer Programs" (SCIP) [1]:

[1] This article is based on a blog post previously published at `https://medium.com/@mathau/learning-to-program-programming-to-learn-c2c3d71d4d1d`

> The computer revolution is a revolution in the way we think and in the way we express what we think. The essence of this change is the emergence of what might best be called procedural epistemology—the study of the structure of knowledge from an imperative point of view, as opposed to the more declarative point of view taken by classical mathematical subjects. Traditional mathematics provides a framework for dealing precisely with notions of "what is." Computation provides a framework for dealing precisely with notions of "how to."

In his "Legacy of CS" essay, he then goes on about pedagogy:

> Traditionally, we try to communicate [...] skills by carefully solving selected problems on a blackboard, explaining our reasoning and organization. We hope that the students can learn by emulation, from our examples. However, the process of induction of a general plan from specific examples does not work very well, so it takes many examples and much hard work on the part of the faculty and students to transfer the skills.

And here comes the most crucial part:

> However, if I can assume that my students are literate in a computer programming language, then I can use programs to communicate ideas about how to solve problems: I can write programs that describe the general technique of solving a class of problems and give that program to the students to read. Such a program is precise and unambiguous—it can be executed by a dumb computer! In a nicely designed computer language a well-written program can be read by students, who will then have a precise description of the general method to guide their understanding. With a readable program and a few well-chosen examples it is much easier to learn the skills. Such intellectual skills are very hard to transfer without the medium of computer programming. Indeed, "a computer language is not just a way of getting a computer to perform operations but rather it is a novel formal medium for expressing ideas about methodology. Thus programs must be written for people to read, and only incidentally for machines to execute." (SCIP)

So, "computer programming" is a kind of advanced pedagogical device! Before we investigate this idea further, let's review the other two reasons for teaching programming to everyone.

3 Reason 1: Programming to Earn

The first reason mentioned in my introduction, the need for more software engineers in industry, has some benefits. There indeed seems to be a need for more professional software engineers, and software engineer is indeed a well regarded and well paid job. And learning to program in school might indeed entice some students to eventually study computer science to become professional software engineers. However, teaching programming to everyone in school, for the *sole* purpose of boosting the number of people who will later study computer science, seems like an inappropriate use of the limited time available in school.

4 Reason 2: Programming to Think

I always wondered why the headmaster of my school told me that studying Latin would teach me to think. That was, until I learned about the Trivium.

The Trivium is an idea from the Middle Ages and defines the foundational education in these times. In a presentation entitled "The Lost Tools of Learning" [9] Dorothy Sayers discussed the idea of the Trivium at Oxford University in 1947:

> The syllabus was divided into two parts: the Trivium and Quadrivium. The second part–the Quadrivium–consisted of "subjects," and need not for the moment concern us. The interesting thing for us is the composition of the Trivium, which preceded the Quadrivium and was the preliminary discipline for it. It consisted of three parts: Grammar, Dialectic, and Rhetoric, in that order.
>
> Now the first thing we notice is that two at any rate of these "subjects" are not what we should call "subjects" at all: they are only methods of dealing with subjects. Grammar, indeed, is a "subject" in the sense that it does mean definitely learning a language–at that period it meant learning Latin. But language itself is simply the medium in which thought is expressed. The whole of the Trivium was, in fact, intended to teach the pupil the proper use of the tools of learning, before he began to apply them to "subjects" at all. First, he learned a language; not just how to order a meal in a foreign language, but the structure of a language, and hence of language itself–what it was, how it was put together, and how it worked. Secondly, he learned how to use language; how to define his terms and make accurate statements; how to construct an argument and how to detect fallacies in argument. Dialectic, that is to say, embraced Logic and Disputation. Thirdly, he learned to express himself in language–how to say what he had to say elegantly and persuasively.

The Trivium taught students to think clearly, and enabled them to learn concrete subjects much more easily. Sayers argues further, that modern education (in 1947) lost track of this idea, and tried to teach students as many subjects as possible, instead of first teaching them how to think (and learn). She argues that teaching the Trivium first would make it dramatically easier and more efficient to learn any number of subjects later.

I wonder whether a modern-day equivalent of the Trivium would contain programming, or maybe, more generally "computational thinking," as one of its non-subject subjects.

In any case, if programming is part of a new Trivium—of the set of subjects that allow people to think clearly—then, obviously, everyone has to learn to program. However, in this article I want to focus on Sussman's somewhat related but different reason for learning to program.

5 Reason 3: Programming to Learn

The third reason for teaching programming to everyone is to enable Sussman's view of using computer programming as a new pedagogical device. This is a very concrete approach: if we are able to program, we can then teach other topics (like biology) by *modeling* the phenomena of those topics using computer programs.

Thus, the reason for **learning to program** is to later enable us to **program to learn**!

Note that I am not proposing that "programming to learn" would completely replace existing pedagogical tools. In fact, programming an executable model of some phenomenon may not be enough to fully understand that phenomenon. For example, if a "programming to learn" activity already starts with a rather complete specification of the phenomenon, a student may be able to implement a working program in a somewhat mechanical way, without profoundly understanding the deeper meaning of the specification or the phenomenon. Nevertheless, I believe that "programming to learn" can greatly augment existing pedagogical approaches.

I came up with the phrase "programming to learn" when reading Sussman's essay. I quickly discovered that others used it long before me. However, their interpretations differ somewhat from Sussman's idea.

5.1 Mendelsohn et al.'s Interpretation

Already in 1990, Mendelsohn et al. brought up that phrase in their book chapter on "Programming Languages in Education: The Search for an Easy Start" [4].

> [...] as new languages have been developed, there has been much discussion of whether the primary aim should be to teach children programming for its own sake, or to use programming in the service of some other end or discipline—"programming to learn, or learning to program."

Mendelsohn et al. look at the two phrases—"programming to learn" and "learning to program"—as two mutually exclusive choices, as two different ways of learning to program. They seem to regard "programming to learn" as a way to learn to program in a meaningful context, where the programming language "acts as a medium for the practicing of specific skills," while they regard "learning to program" as a way to learn to program in a decontextualized way, where the "programming language is above all a new language to be learnt."

In Sussman's view, "programming to learn" does not seem to *include* "learning to program." That is, students first learn to program, and, after that, they can use the gained programming skills to use programming to learn.

5.2 Miller's Interpretation

In his 2004 dissertation, "Promoting computer literacy through programming Python," [5] John Alexander Miller puts "programming to learn" in a more general context:

- "learning to read" enables students to "read to learn"
- "learning to write" enables students to "write to learn"
- "learning computer literacy" enables students to "use computer literacy to learn"
- "learning to program" enables students to "program to learn"

Miller's work connects "programming to think" and "programming to learn." On one hand, he considers programming as a cross-cutting part of a new Trivium, proposing to replace Latin with Python. This is more about acquiring computational thinking skills than learning to program to enable Sussman-style learning experiences. On the other hand, Miller highlights the power of programs as "executable notations" and concludes with:

> [...] integrating programming into curricular activities may significantly alter *what* knowledge becomes important to learn in many of the traditional subject areas, as well as *how* that knowledge is learned.

Sussman's idea is exactly about using programming to *alter how knowledge is learned*.

5.3 Resnick's Interpretation

In 2012, Mitchel Resnick, Professor of Learning Research at the MIT Media Lab, gave a TED talk with the title "Let's teach kids to code" [7], where he brought up the phrase "Code to Learn." He further elaborated the idea in a 2013 EdSurge post titled "Learn to Code, Code to Learn" [8]. He argues that coding helps you to learn how to: experiment with and communicate new ideas, take complex ideas and break them down into simpler parts, collaborate with other people on your projects, find and fix bugs when things go wrong, and keep persistent and persevere in the face of frustration.

Resnick's interpretation of "code to learn" seems to align with the second reason: learning to code to help students to become more literate, critical thinkers and creators. The skills he enumerates are helpful in many other contexts just like the "thinking skills" my Latin teacher instilled are helpful in other contexts. Resnick's interpretation does not, however, directly and completely address the third reason: learning to code to provide students with the means (coding) so they can learn in the entirely new way described by Sussman.

5.4 Wenger's Interpretation

In a 2012 blog post "Learning to Program, Programming to Learn" [10], Albert Wenger writes:

> Programming is "teaching" the computer how to do something. If you can't teach it to the computer you have probably not completely understood it. Hence the "programming to learn" in the subject line of this post.

He enumerates a set of school subjects in which he sees programming helping with learning: History (programming animated maps and historical timelines), English (programming word games), Music (programming music, sound, and visualizations thereof), Science (programming simulations), and Math (programmatically illustrate the number line, connect algebra and geometry).

Wenger's interpretation is very close to Sussman's. While Sussman describes the teacher writing a program for students to read, Wenger talks of the students writing the program themselves. Wenger's approach thus represents a more active approach to learning and thus is more in line with modern pedagogy. Another difference is that Sussman focused on college-level courses, while Wenger discussed using programming in school.

5.5 Guzdial's Interpretation

In his 2015 book on "Learner-centered design of computing education: research on computing for everyone" [3], Mark Guzdial, a computing education professor at the Georgia Institute of Technology, discusses a list of potential reasons for why everyone should learn computing: jobs, learn about their world, computational thinking, computational literacy, productivity, and to broaden participation.

Guzdial's book is a treasure trove of information about studies on computing education for everyone. Overall, Guzdial finds that what I call "computing to think" is less promising, because there is no evidence that one can teach general, transferable problem-solving or thinking skills by teaching programming. However, he argues and cites evidence that programming can be helpful in learning subjects like mathematics, science, or engineering:

> Writing a program to solve an engineering problem may give students new insights into the engineering problem. Writing a program to express an idea can transform how the programmer thinks about that idea. This is not about transfer. This is about the power of using a new medium, of applying computing to new domains.

This very much corresponds to Sussman's "computing to learn" perspective.

6 Programming to Learn What?

I find that idea of "programming to learn," in Sussman's sense, extremely powerful! And I realize that I and many of my colleagues have been applying that idea in our courses for a long time: we have students implement a concept as a program just so they will profoundly understand that concept.

Given that I teach concepts in computer science, the concepts I teach often are methodological and amenable to modeling and implementation in code. For example, I taught computer architecture by having students implement a simple micro-architectural simulator. Or I teach version control by having students implement a simplified simulator of Git. And who has ever taught compiler optimization without having students implement at least parts of a compiler?

I wonder where the boundaries of this "programming to learn" approach lie. Can you teach statistics this way, and how well can you do so? Biology? Economy? Psychology? Sociology? How about history, or natural languages? Carpentry? Plumbing? Or... art?

I bet the boundaries lie far beyond computer science. And I bet few non-computer-scientists actually use a "programming to learn" approach. Sometimes because they never learned to program. Other times because their students can't program.

7 Do Not Neglect Learning to Program

Programming to learn might be the most important reason for learning to program. However, in order to program to learn, we *do* have to learn to program. If we single-mindedly focus on "programming to learn," on how students can *use* programming to learn other topics, and if programming disappears—or is never introduced—as a curricular subject, then we risk that students will not learn the foundational concepts underlying "programming." As a result, they will not be able to program to learn either.

This risk is not negligible. A similar phenomenon happened in Switzerland [2], when informatics was eliminated as a subject from the high school curricula, and instead was integrated into the various subjects in which it could be applied. The result was that students did not really learn programming anymore, but only learned how to use computers as end-users.

8 Conclusions

In order to program to learn, you first have to learn to program. So we have to teach coding, or programming, to everyone! Not because they need to become professional programmers. Not even because it improves their critical thinking skills. But simply because it opens up a new, potentially powerful way of learning. By reading and writing code.

Acknowledgements

In addition to thanking the authors of the inspiring works I cited, I would like to thank all the colleagues who influenced my thinking about these ideas. Special thanks go to Judi Fusco, Patti Schank, Shuchi Grover, and Jeremy Roschelle at SRI International for the many insightful discussions about the learning sciences and computer science education. However, the interpretations and opinions presented here are my own and may not necessarily represent those of my colleagues.

References

[1] H. Abelson and G. J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, MA, USA, 2nd edition, 1996.

[2] W. Gander. Informatics – new basic subject. In *Bulletin of EATCS*, number 116. European Association for Theoretical Computer Science, June 2015.

[3] M. Guzdial. *Learner-centered design of computing education: research on computing for everyone*. Morgan & Claypool Publishers, Nov. 2015.

[4] P. Mendelsohn, T. R. G. Green, and B. Paul. Programming languages in education: The search for an easy start. In J.-M. Hoc, T. R. G. Green, D. Gilmore, and R. Samway, editors, *Psychology of Programming*, chapter 2.5, pages 175–200. Academic Press, London, 1990.

[5] J. A. Miller. *Promoting Computer Literacy Through Programming Python*. PhD thesis, University of Michigan, Ann Arbor, MI, USA, 2004. AAI3122001.

[6] National Research Council. *Computer Science: Reflections on the Field, Reflections from the Field*. The National Academies Press, Washington, DC, 2004.

[7] M. Resnick. Let's teach kids to code. TED Talks. `http://www.ted.com/talks/mitch_resnick_let_s_teach_kids_to_code`, Nov. 2012.

[8] M. Resnick. Learn to code, code to learn. EdSurge. `https://www.edsurge.com/news/2013-05-08-learn-to-code-code-to-learn`, May 2013.

[9] D. L. Sayers. The lost tools of learning. Essay presented at Oxford University. `http://www.gbt.org/text/sayers.html`, 1947.

[10] A. Wenger. Learning to program, programming to learn. Blog post. `http://continuations.com/post/36062400780/learning-to-program-programming-to-learn`, Nov. 2012.