Coding in the curriculum: Fad or foundational?

Emeritus Professor Leon Sterling
Swinburne University of Technology, Victoria

Professor Leon Sterling received a BSc (Hons) from the University of Melbourne and a PhD in Pure Mathematics from the Australian National University. He has worked at universities in the UK, Israel, the US, and Australia. His teaching and research specialties are artificial intelligence, software engineering, and logic programming. Leon had a range of roles for 15 years at the University of Melbourne, including Head of the Department of Computer Science and Software Engineering, Professor of Software Innovation and Engineering, and Director of e-Research. He served at Swinburne University of Technology as Dean of Information and Communication Technologies from 2010–2013 and as Vice Chancellor (Digital Frontiers) from 2014–2015. He served as President of the Australian Council of Deans of ICT from 2012–2014.

Introduction

There has been an explosion in mobile devices over the past decade, with the associated issue of developing the skilled workforce needed to write the apps that run on the devices. This has been a significant factor in highlighting what is taught in schools – STEM education in particular. For schools, technology – the 'T' in STEM – is primarily digital technology.

This paper concerns what should be taught in digital technology, and specifically the role of computer coding. We take it for granted that computers are now essential in schools, and students need basic computer literacy skills. Pleasingly, basic computer literacy is a separate curriculum item from digital technology, and is not the subject of this paper. Note that naming the discipline underlying digital technology has been challenging. Computer science, informatics and computational thinking have all been suggested and used, with advocates and detractors for each name.

Computer scientists prefer the term computational thinking, a position advocated 10 years ago by Jeanette Wing (2006), with wide adoption. According to Wing, 'computational thinking involves solving problems, designing systems, and understanding human behaviour, by drawing on the concepts fundamental to computer science.' Much material has been developed to teach computational thinking, with Computer Science Unplugged (Bell et al., 2015) an influential and representative resource.

Abstract

There has been an unprecedented push to revitalise interest in STEM education. Much of the discussion of the 'T' in STEM education has centred around whether coding should be a central element of school education. This paper investigates arguments for and against 'coding in the curriculum'. No sensible person thinks that teaching coding in the classroom will produce master programmers, any more than teaching music in the school curriculum will produce master musicians. However, the teaching of music can encourage some students to become musicians, and the same would be true for coding. The issue is more what concepts are addressed in teaching coding, and how essential they are for engendering an understanding of the digital world around us, and improving productivity and innovation, for which ICT skills and capability are essential.

Grover and Pea (2013) provide a systematic review of progress in implementing computational thinking in the curriculum for the six years immediately following Wing's influential position paper; they note the Committee for the Workshops on Computational Thinking run by the National Research Council (NRC) in the United States of America, with associated reports (NRC, 2011). Grover and Pea take an educational research perspective and are largely positive.

Where should computational thinking be placed in the curriculum, and what topics, if any, should it displace? My personal belief is that computer science is the new applied mathematics. Just as mathematics applied itself to the physical world, explaining mechanics and electro-magnetism, we are currently applying mathematics to understanding data, information and knowledge. Thus, computational thinking has a role in mathematics curriculum, and also in a science curriculum where insights provided by data add to our scientific knowledge. Indeed, software is essential to many physical devices like telescopes and microscopes, and should be explained as such to students. If students work in advanced scientific fields, they will be interpreting the results of programs and they need to understand how computers operate. Admittedly, there is a lack of agreement on whether computational thinking should ultimately be incorporated into education as a general subject, a discipline-specific topic, or a multidisciplinary topic (NRC, 2011).

Teaching computer programming in schools

Rather than focus on computational thinking in this paper, however, I want to discuss the more contentious issue of teaching computer programming in schools. As discussed in Webb et al. (2016):

> The distinction between computational thinking and programming is subtle; in principle computational thinking does not require programming at all, although in practice, representing a solution to a problem as a program provides a perfect way to evaluate the solution, as the computer will execute the instructions to the letter, forcing the student to refine their solution so that it is very precise.

The phrase 'coding in the curriculum' seems to be the current preferred option to programming, presumably partly because it is catchy. Note that much of the discussion seems to be happening in social media and blogs rather than the academic literature.

A case for students learning coding is well-made by Professor Mitchel Resnick from MIT's Media Lab in his 2012 TED talk (Resnick, 2012). Resnick is the designer of Scratch (Resnick et al., 2009), the leading language for teaching coding to primary students, which is also used for teaching secondary students. Scratch is a fun and engaging collaborative environment that has been popular and successful. Resnick's argument centres on the positive design skills that students gain from undertaking a project with Scratch.

What are the benefits of teaching children to code from an early age? In my opinion, what is important is twofold: the thinking engendered by coding, and an appreciation of what computers can and cannot do, laying the groundwork for what they may do in the future. A typical argument in social media is contained in a blog post (Tufts, 2016) that lists seven benefits. The benefits fall loosely into three groups: teaching children general problem-solving and design skills – essentially the arguments for computational thinking; introducing the students to the environments they will be using in the future; and encouraging more students to take up careers in coding, with benefit to society and the workforce.

There is merit in students having positive experiences with environments they are likely to meet later in life. Scratch and other environments have communities within them that encourage and enable code sharing, cooperating and mentoring. Many children have tablets and other technology, and experience with coding brings the home and the classroom closer together. However, experience with the tablet environment is essentially an argument for digital literacy.

The argument on teaching coding because society needs more professional coders is a stretch. We teach music and sport in schools because of the inherent value in music and sport rather than because we need more professional musicians and sportspeople. Incidentally, programmers are often the sharpest critics of teaching coding, as they think it detracts from the coding profession. One coding class at school does not make a professional programmer. However, it can identify talent and interest.

Pedagogy and positive outcomes

I would like to address several potential objections to placing coding in the curriculum. The first argument is that teaching coding does not come from an adequate pedagogical basis. In my opinion, the pedagogy is under control. There is consensus that Scratch works well. Concepts underlying Scratch are drawn from a tradition of research dating back to Seymour Papert in the 1960s, 1970s and 1980s. The key features of using a block-based programming language, avoiding children having to worry about minor syntax issues, being able to rapidly see the results of executing programs, and being able to draw on a rich library of multimedia are all significant.

And Scratch is not the only option. In recent years, there has been an increase in the number of programming environments that are freely available for use by novice programmers, particularly children and young people (Good, 2011). There is much training material of high quality, including Codecademy (https://www.codecademy.com); Code Club in the UK and Australia (https://www.codeclub.org.uk and http://www.codeclubau.org) and elsewhere; Code.org (http://code.org); and commercial providers such as Tynker(https://www.tynker.com), to name a few. To some extent, market forces have ensured suitable pedagogy.

The second argument is that there is no evidence base establishing that coding is beneficial. That is not correct, but the evidence is primarily anecdotal, rather than from random experimental trials. A typical effort to introduce programming to primary schoolchildren, using Scratch, is described in Wilson and Moffat (2010). From the abstract:

[W]e used Scratch to teach some elementary programming to young children (eight years old) in their ICT class, for eight lessons in all. Data were recorded to measure any cognitive progress of the pupils, and any affective impact that the lessons had on them. The children were soon able to write elementary programs, and moreover evidently had a lot of fun doing so. An interview with their teacher showed that some of the pupils did surprisingly well, beyond all expectations.

As Wilson and Moffat comment:

While the cognitive progress is moderate, the main advantage to Scratch in this study seems to be that its enjoyability makes learning how to program a positive experience, contrary to the frustration and anxiety that so often seems to characterise the usual learning experience.

While a rigorous trial is preferable to anecdotal evidence, the difficulties of running a rigorous experiment should be acknowledged. It is difficult to justify running control groups where some students gain the benefit of learning coding and others do not. It is hard to have comparable teaching. The passion and skills of the teacher are currently influential on how successful classes are in teaching coding. As languages are rapidly evolving, it is not clear what standards should be used for evaluating trials of technology. There should be active discussions about what the evidence should be. There are active discussions about assessment, as noted by Grover and Pea (2013) and others.

The next potential objection is that the push for coding is primarily about vested interests. Indeed, vested interests influenced the push for computers in the classroom. Negative experiences in introducing computers in the classroom might deter some people from trying to teach coding. Large multinational companies like to lock schools into their particular products. However, advocating for teaching coding in the curriculum is different to advocating for computers in the classroom. The drivers for coding are public interest groups as well as vendors, and there are quality resources that are free and open-source. Nonetheless, there is considerable scope for research on distinguishing between claims of competing products and environments for teaching coding.

It is significant that there is much collaboration happening between academic interests and industry. For example, two initiatives aimed at introducing computing into schools, CS4HS (http://www.cs4hs.com) and the Code.org Advocacy Coalition, represent collaboration between academia, national bodies, and industry leaders such as Microsoft and Google. The Computer Science Teachers Association’s Model Curriculum for K–12 Computer Science, supported by the Association for Computing Machinery (the largest computing professional association) provides suggestions to help engage and motivate students (https://csta.acm.org/Curriculum/sub/CurrResources.html). Google’s Exploring Computational Thinking website (http://www.google.com/edu/computational-thinking) has a wealth of links to web resources.

Another complaint is that current popular Scratch-like environments for students are too limited to learn the important concepts in programming. That concern is being addressed. Snap! (http://snap.berkeley.edu) is a well-designed extension which is used in Algorithmics, the Victorian VCE subject. Other environments facilitate transition from a block-based language to the text-based syntax used in industry. For example, Code.org facilitates transition from a Scratch-like block-based language to the JavaScript language.

Coding in the curriculum

Let us reconsider the place of coding in the curriculum. Is there a compelling rationale for all children, including those who allege no interest in pursuing STEM careers, to learn coding in school? Space can be made in the curriculum by connecting coding to mathematics and science lessons. Computing examples and well-designed exercises can highlight the relevance of maths and science. Recognising faces, translation between languages, and searching in large collections can all be explained in terms of data, and provide practical and interesting experiences for using coding and scientific methods. Computing projects can easily be structured to give students experience with important generic skills such as persistence, collaboration and communication. Overall, I believe that coding is foundational.

What about year level? The Australian curriculum for digital technology sets objectives for each year level from

K–10. The approach is ambitious, but well structured. Coding should be a key component of meeting the digital technology curriculum objectives.

There has been some discussion that learning a computer language is like learning a foreign language. Indeed, earlier this year, a bill was approved in the Florida senate allowing high school students to take computer coding classes in place of foreign language requirements. That is not a position I support. Supporting science and mathematics is a better place for coding in the school curriculum than replacing the teaching of second languages. Using language is about communicating with people and recognising the culture from which the language emanates. Communication between people is fundamentally different from communicating between human and computer.

Worldwide there is momentum behind teaching coding. Many countries are experimenting with including coding in the school curriculum. Last year, the Australian Labor Party issued a platform entitled 'Coding in Every Australian School'. Webb et al. (2016) discuss vignettes from five countries: the United Kingdom, New Zealand, Australia, Israel, and Poland, where programming is in the curriculum. Much can be learned from these experiences.

One concern is that teachers may not have the skills to teach computer coding correctly. Resources are being prepared. In May, the Australian Department of Education awarded a project after a tender for National Computing Challenges for Year 5 & 7 and Cracking the Code, which are helping with teacher and student resources.

Competitions are growing. The ACER Australian STEM Video Game Challenge (https://www.stemgames.org.au) introduced in 2014 has had excellent uptake. Learning to code games is fun and exciting, and can spark interest in digital technology.

Summary

In summary, what have we learned so far about teaching coding in the curriculum? Plenty of experimentation is happening. Projects introducing coding through Scratch or similar positive environments are largely successful. Teaching computing can be made to be engaging, though perhaps not to everybody. Being able to see the results of executing the code immediately is essential. Curriculum material is being developed. The lack of resources for teachers is being addressed, though there is a challenge to produce resources in time. Note that the block-based languages are more accessible to teachers, just as they are for students, such that many more teachers are able to create or modify resources.

My personal opinion is that coding should be taught in all schools. While it is not necessary nor realistic that all students become coders, it is important that they appreciate what computers do and how they do it. The best way I know of conveying the understanding is by having students code. Some students struggle to learn to code. However, without attempting to code, something essential is missing.

References

Barr, V. & Stephenson, C. (2011). Bringing computational thinking to K–12: What is involved and what is the role of the computer science education community? *ACM Inroads, 2*(1), 48–54.

Bell T., Andreae, P. & Robins A. (2014). A Case Study of the Introduction of Computer Science in NZ Schools. *ACM Transactions on Computing Education, 14*(2), 1-31. http://dl.acm.org/citation.cfm?doi: 2642651.2602485

Bell, T., Witten, I. H. & Fellows M. (2015). *CS Unplugged, An enrichment and extension programme for primary-aged students*. Adapted for classroom use by Robyn Adams and Jane McKenzie. 2015 Revision by Sam Jarman.

Connell, A., Hramiak, A. & Edwards, A. (2015). *A Practical Guide to Teaching Computing and ICT in the Secondary School* (2nd edition). Abingdon, United Kingdom: Routledge.

Good, J. (2011). Learners at the Wheel: Novice Programming Environments Come of Age. *International Journal of People-Oriented Programming, 1*(1), 1–24.

Grover, S. & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher, 42*(1), 38–43.

Moreno-León, J., Robles, G. & Román-González, M. (2015). Dr. Scratch: Automatic Analysis of Scratch Projects to Assess and Foster Computational Thinking. *Revista de Educación a Distancia*, No. 46. http://www.um.es/ead/red/46/moreno_robles.pdf

National Research Council. (2011). *Committee for the Workshops on Computational Thinking: Report of a workshop of pedagogical aspects of computational thinking*. Washington, DC: National Academies Press.

Resnick, M. (2012). *Let's Teach Kids to Code*. https://www.ted.com/talks/mitch_resnick_let_s_teach_kids_to_code?language=en#t-214742

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. &

Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM 52*(11) 60–67 http://dx.doi.org/10.1145/1592761.1592779

Rutstein, D. W. & Grover, S. (2016). *Measuring Student Learning about Computing*. https://www.sri.com/blog/measuring-student-learning-about-computing

Shein, E. (2014). Should everybody learn to code? *Communications of the ACM, 57*(2), 16–18.

Tufts, P. (2016). *7 Lifetime Benefits of Teaching Coding Games for Kids to Your Kids*. http://learntocode.biz/7-lifetime-benefits-of-teaching-coding-games-for-kids-to-your-kids

Webb, M., Davis, N., Bell, T., Katz, Y. J., Reynolds, N., Chambers, D. P. & Sysło, M. M. (2016). Computer science in K–12 school curricula of the 21st century: Why, what and when? *Education and Information Technologies*, (pp. 1–24).

Wilson, A. & Moffat, D. (2010) *Evaluating Scratch to introduce younger schoolchildren to programming*. Proceedings of the 22nd Annual Workshop of the Psychology of Programming Interest Group, Cambridge, UK. http://scratched.gse.harvard.edu/sites/default/files/wilson-moffat-ppig2010-final.pdf

Wilson, C. & Guzdial, M. (2010). How to make progress in computing education. *Communications of the ACM, 53*(5), 35–37.

Wing, J. M. (2006). Computational Thinking*.* *Communications of the ACM, 49*(3), 33–35.