Is CS Really for Kids?

Observations of a CS teacher related to K-12 CS education

Author: Abhay B. Joshi

Introduction

The idea of using *computer programming as a medium for learning* was first proposed by Seymour Papert in his ground-breaking book "Mindstorms" back in the 1980s. Dr. Papert also designed the Logo programming language specifically for this purpose. For a variety of reasons this idea remained an esoteric one and did not become main-stream until almost the turn of the century. The last decade and half has seen a dramatic change in this situation. Today, no one questions the idea anymore. The benefits of learning programming and *computer science* concepts well before college – even in elementary grades – are well-understood. Everyone from the President of the United States to CEOs of major tech companies to school administrators are talking about teaching *computational thinking* to children and young adults. The focus is now on *how* to implement this idea in schools.

Here is a list of some of the amazing things that happen when children engage in computer programming:

- Children become *active* and *creative* learners, because they explore ideas through a hands-on activity with an infinitely powerful tool.
- They learn to think about and analyze *their own thinking*, because that is the only way to program computers.
- They learn to solve complex problems by breaking them into smaller sub-problems.
- They learn a new way of thinking (called "computational" thinking).
- In the world of programming, answers are not simply "right" or "wrong"; this prepares a child's mindset for real-life problems.
- Children's learning processes are transformed from *acquiring facts* to *thinking creatively and analytically*.

Current State of Affairs

With the above benefits in mind and with the goal of encouraging kids to take up STEM careers later in college, a lot of people are hard at work in universities, the computer industry, schools, and various communities. Their efforts are focused on:

- Encouraging young students to take up coding, e.g. the Hour of Code.
- Training/coaching CS teachers for K-12 levels, e.g. UC Berkeley's BJC course
- Developing friendly programming systems, e.g. Scratch, Blockly, AppInventor, etc.
- Creating on-line learning material and courses for teachers as well as students, e.g. Scratch courses on EdX.org and Coursera, and courses on Code.org.

I feel there are a couple of problems in all this exciting activity that need attention of parents, educators, and thought leaders.

Problem # 1: Irrational Rush?

Parents have caught on with the idea of introducing their kids to programming and have started sending their kids to programming classes or clubs at very young ages – even 1st grade. And several businesses and non-profit groups have sprung up to enthusiastically cater to this new class of customers. There are several elementary schools in the Seattle area, for example, where coding classes are offered as after-school activity, which are conducted by external vendors or non-profit organizations.

Simply stated, the problem, I think, is that *we are exposing children to computer programming rather too early*!

While there might be a small percentage of whiz-kids who can learn and enjoy programming at a very young age, a large majority cannot. And worse, they might get turned off even before we have had the proper opportunity to expose them to this wonderful idea.

Programming is not just about learning commands, but it is also about understanding the underlying CS concepts; it is also the art and science of putting these two together to solve interesting problems, to design interesting programs. The most important skills required to enjoy programming at any level are:

- Ability for abstract thinking (ability to generalize, create mental models, etc.)
- Ability to work systematically (the mental discipline of building things piece by piece)
- Ability to concentrate (and not get distracted and demotivated easily)
- Adequate knowledge of areas in which to apply the programming skills (e.g. geometry if you want to draw shapes, basic mechanics if you want to design action games, etc.)

Of course, one may argue (and I agree completely) that these are exactly the type of skills that get *developed* when children start doing programming!

So, how can one demand that they be a *prerequisite* to write programs?

Yes, a kid *can* start programming without any of these skills, but, he/she can't go very far (which also creates another problem as I have narrated below in the "Choice of language" section). For programs that are meaningful and also fun, the average child needs the foundation of primary education and some elements of the above-mentioned skills. These skills get *refined* when children get involved in interesting programming projects. I feel that the joy of programming increases exponentially as kids build these required skills. In other words, my point is that the effectiveness of learning programming at the post-primary age is dramatically higher than it is at the primary school age.

Problem # 2: Choice of Language

The other problem in our over-enthusiastic rush towards K-12 CS education is the misguided choice of programming languages. Yes, there may be some 4th grade whiz-kids who can pick up JavaScript, but, such kids are exceptions. The number one reason why kids (and even adults) get turned off by CS is the text-based interfaces, obtuse syntax, and un-friendly development environments of conventional languages such as Java, JavaScript, Python, C, C# etc. If you see for yourself the peculiar formatting notions, unfriendly compiler warnings and errors, and the hours of frustration that goes into getting even the simplest piece of code working, you will appreciate kids' unfavorable reaction to these languages as their first introduction to programming.

But, the good news is that *language* is not a problem anymore! The days of starting your computer programming career with the likes of Python and C are long gone!

Ok, so, how do we fix the problems listed above? Here is what I suggest.

The Right Approach

We should introduce computational thinking after primary school, i.e. 5th grade or later, and with languages like Scratch.

I don't think waiting till the 5th grade should be a problem at all. What is the rush anyway? There are four long years of middle school during which a child can get introduced to computational thinking skills and then decide whether to pursue them more rigorously at the high school level (by pursuing STEM options and/or AP-CS type of courses), or to carry this knowledge with him/her for something totally different.

Besides, there is so much else – and probably more important than computational thinking – to learn at the primary school age! Sports, music, human languages, outdoor activities, social interaction, … The list can go on of exciting things that pre-middle-school kids can and should be learning instead of jumping into computer programming. I am no neuro-scientist or psychologist, but based on my limited knowledge and experience our young kids badly need these other things more than intimate familiarity with computer gadgets.

And finally, if you still want to get your kids started early, it is best to employ non-programming techniques such as, outdoor games or group activities (see [2] in References) to give young kids a flavor of computational thinking.

Enter Scratch: an underrated language

Nowadays there are quite a few friendly and exciting programming languages, the best, in my opinion, being Scratch. Scratch was designed by no dummies either. It was designed at the famous Media Labs at MIT, which pioneered the whole idea of teaching programming to kids in the early 1980s through their first kid-friendly language called Logo.

Scratch requires very little typing. All commands appear as Lego blocks which the programmer has to assemble together by dragging and dropping into the "script area". The stress of dealing with "syntax errors" is thus minimized. You can design highly interactive and visually appealing programs easily in Scratch. It also embeds many of the standard CS concepts and even a few advanced concepts like *concurrency* that are not easily accessible in other languages. Besides being backed by a solid institute like MIT, Scratch is supported by an incredible amount of learning resources: a website that hosts millions of sample programs, a host of online concept videos, a special website for educators and teachers, and so on. A number of online courses also exist for anyone who is interested in formal training.

And yet, Scratch's simplicity is deceiving. Parents often come to me and say, "Ok, my 4th grade son has finished a course in Scratch. What should he do next?" The implication is that the boy has exhausted the capabilities of Scratch and is ready for something more challenging! Companies and businesses that exploit Scratch programming classes for 1st or 2nd grades indeed exhaust the capabilities of Scratch very quickly. Because those kids can't do much programming anyway! So, you can't really blame their parents for asking for something new after two back-to-back courses of Scratch (deceptively titled “Basic” and “Advanced”!).

The truth is Scratch is capable of challenging even a college student. As I have mentioned, programming is not just about a set of commands and concepts. It is about putting them together to solve problems. And Scratch commands coupled with CS concepts can be put to use for problems that are quite challenging. I have taught Scratch to more than 800 7th and 8th graders since 2009 and I have never faced the problem of how to challenge them – even the brightest among them. In fact, I have never been able to truly reach the limits of Scratch with these students.

The capacity of Scratch to engage students and get them interested in the process of thinking, analyzing, debugging their own thought process, coding again and again without getting tired or frustrated, sharing enthusiastically their working programs with their friends, is beyond comparison to any other language.

I do not think there is a better choice of language (than Scratch) for introduction to computer science principles and computer programming at any age.

The problem, as we discussed above, is the practice of introducing Scratch (and programming itself) too early. And the other problem is not taking the trouble to get deeper into Scratch, beyond its kiddy-looking sprites and Lego blocks, to truly tap into its computational potential. As a teacher, whenever I took that trouble I have reaped wonderful dividends for myself and for my middle school students.

Advanced Scratch Programs

I have listed below a few examples of Scratch programs that could easily challenge middle-school or even high-school students. See [3] in References for links to actual programs that you can run.

- Tower of Hanoi: This program provides the framework to solve this famous puzzle, both in user and automatic modes.
- Recursive designs: Using the idea of recursion, this program draws designs such as snowflakes, the Sierpinski triangle, spiral etc.
- Arithmetic game: This game challenges young kids to practice their arithmetic skills. It presents simple equations and asks the user to guess the arithmetic operation.
- Card memory game: This program implements the popular game in which users try to pick pairs of matching picture cards.
- Optical reflection: This program simulates the law of light reflection: the angle of incidence is equal to the angle of reflection on a flat surface.

Conclusion

The benefits of exposing kids to computational thinking well before college are beyond any doubt. The effort of most educators, parents, and related stakeholders is now focused on empowering teachers and students to learn computer science. This article lists, based on actual field experience of 8 years, two possible flaws in the CS strategy of some of these stakeholders. One has to do with introducing CS too early – i.e. at primary or pre-primary levels. And the other has to do with improper choice of programming language. The article goes on to suggest that the most benefit can be had if we introduce CS after primary school education. The article also contends that Scratch is a perfect programming language to afford our students a wide range of entertaining, challenging, and rich programming experiences.

References

1. There are several books and papers on the subject of "computational thinking for children and young adults". If you would like to get a quick overview of this subject, please check out this 2-page handout at: http://scratched.gse.harvard.edu/resources/handout-learning-through-programming
2. http://csunplugged.org/: Activities that teach computational thinking without the use of computers.
3. Examples of challenging Scratch programs:
    a. Tower of Hanoi: https://scratch.mit.edu/projects/87006602/
    b. Recursive designs: https://scratch.mit.edu/projects/105188147/
    c. Arithmetic game: https://scratch.mit.edu/projects/102812152/
    d. Card memory game: https://scratch.mit.edu/projects/106144703/
    e. Optical reflection: https://scratch.mit.edu/projects/101456673/

Author's Background

Abhay Joshi is a CS teacher and he has been teaching computer programming to young kids since 2007. He has taught kids of grades 2nd to 10th. He has taught Logo, Scratch, Alice, and C,

and he personally knows languages such as Java, Python, and C++ due to his earlier software industry experience (of about 20 years). His most successful and enjoyable teaching experience has been with middle-school kids (7th and 8th grades) whom he teaches Scratch. In the constructivist tradition of Piaget and Papert, Abhay views Computer Programming not just as a useful skill to build careers, but also as a "powerful medium for learning" in which students actively engage in a creative, entertaining, and intellectually challenging pursuit. Abhay has written books on Logo Programming (2011) and Scratch Programming (2016).

Abhay is not affiliated with MIT or any of its Scratch teams. Abhay is a freelance teacher and teaches CS as a non-profit activity.