Roadmap to learn CS: a guide for parents

This article proposes a roadmap for a school child to develop computational skills.

Age-group:

We should introduce *computer programming* after primary school, i.e. 5th grade or later (about age 10+).

I don't think waiting till the 5th grade should be a problem at all. There are four long years of **middle school** (5th to 8th) during which a child can get introduced to computational thinking skills and then decide whether to pursue them more rigorously (i.e. more towards a CS career) at the **high school** level (9th thru 12th), or to carry this knowledge with him/her for something totally different.

Look up my article (abhayjoshi.net/spark/articles/IsCSforKids.pdf) that discusses the dangers of starting your child on *Computer Programming* too soon.

What about till then?

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 neuroscientist 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.

Having said that, if you really want to get your kids started early with computational thinking, because they show great interest and/or potential, consider the following:

1. Non-programming techniques such as, outdoor games or group activities to give young kids a flavor of computational thinking:
    a. https://cspathshala.org/
    b. http://csunplugged.org
2. Scratch Junior (www.scratchjr.org)
3. Challenging board games and/or computer games (games that promote thinking, creativity, memory development etc.)

The Truth about CS:

The truth is that CS is really just a skill, a way of solving problems, a way of thinking, an assistant in the creative process. It is a tool, like a hammer or a screw-driver. Yes, it is indeed the most powerful tool man has ever invented. But, the point is, you need to know both *where* and *how* to use this tool. A lot of the so-called "computer professionals" only know the second part – that too in a narrow sense. You have to tell them where to use it. It's like the guy has the hammer and says, "Tell me where to hit!" Such people should be called "coders" and not programmers.

It may be somewhat fun and interesting just to bang the hammer where someone asks you to, but, it is a lot more fun and satisfying to be able to have your own ideas of where to apply the hammer. In fact, the future is not far off when these "hammer-carrying" guys (or "coders") will be obsolete and out of work, because everyone will be carrying their own personal hammer and will know how to use it when required.

Coming back to the point, CS skill alone is not sufficient. You will also need to be good at something else – some field such as art, architecture, biology, engineering, politics, law, anything! So, your child must develop their own specialty – fields of knowledge, expertise, a creative talent. As a parent, you must expose your child to languages, music, mathematics, sports, science, and so on. You must let your child discover through exploration their talent, their inspiration, their calling.

Thus, computer skills – including CS skills – must not receive all your focus in exclusion to these very important considerations, even if, your child wants to do nothing else but sit in front of the computer or hold a gadget in their hands. You must use all your ingenuity to prod the child in all the other directions as well.

Scratch: the best beginning

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* and *object-oriented thinking* 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 high school 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.

The Challenge Parents Face:

A lot of parents want their kids to learn Computer Programming, but have several difficulties as indicated by their concerns/questions below:

1. I do not have the time to teach them.
2. I am not a programmer or do not have computer/programming background.
3. I can, if required, spend some time to teach kids these concepts by reading books or videos.
4. My spouse can spend time too, but s/he does not have the CS background.
5. My kids don't want me to teach anything, because they think it's a part of studies or schoolwork.

Formal Courses:

If you can find a computer club or similar activity (commercial or non-profit) that conducts Scratch Programming courses, I would highly recommend that you verify their credentials and put your child in their courses. Make sure that the coursework is gently paced and allows the child to explore their creativity.

A Viable Alternative:

The truth is for most parents there is no such access to good Scratch learning facilities for their children. Here is a proposal that addresses all their concerns mentioned above and has a great chance of success.

(1) **Kids need some handholding**: They can and do learn independently but need someone to answer questions, give directions (like what is next), and bring some discipline to the learning process (regular activity, encouragement, etc.). So, some adult – one of the parents or some uncle/auntie – must commit some time with the kid. The time commitment is about 2 to 4 hours per week at a minimum. More time is welcome but not necessary. Let us call this person the "facilitator".

(2) **Programming not required**: It is ideal if the facilitator already knows computational thinking (i.e. programming), but it is not necessary. If they are new to programming, they must at least be willing to learn along with the child. The good news is, they can learn as easily as the child can. They can learn together!

(3) **It is best to learn as a group**: There should be at least 2-3 kids (friends) who should join this learning effort. Kids learn best in a group because they can collaborate, share ideas, and have fun – which is an essential element of learning programming. Just one parent teaching his/her child rarely works. As mentioned above, kids think it is some boring school stuff.

(4) **The facilitator is not a silent spectator**: They must do everything the kids are doing, i.e. reading the material together, writing the programs, etc. I can guarantee that if there is sufficient motivation, the facilitator would also have a lot of fun.

(5) **The learning process**: There is a set of books on "Scratch programming" written by the author of this article that are exactly the kind of resource that can come handy in such circumstance. Here is how the learning plan would work.

1. "**Learn CS Concepts with Scratch**" (abhayjoshi.net/spark/scratch/bscratch.pdf) is the first book in that journey. It is written in the format of a class-room course. The group just needs to follow each “Unit” and the chapters in the given sequence and do the programming exercises. The book provides all kind of help: links to solutions, links to demo videos of the solutions, etc.

2. "**Pen Art in Scratch Programming**" (abhayjoshi.net/spark/scratch/pscratch.pdf) is the next book they follow. This book focuses on "drawing pictures and graphics" using Scratch. Once again, it is like a classroom course and comes with all kinds of help.

3. "**Advanced Scratch Programming**" (abhayjoshi.net/spark/scratch/ascratch.pdf) is the final step, which contains a series of challenging and interesting Scratch projects. Each project is explained in detail. Students are encouraged to write the programs on their own, but they can get hints/help/solution whenever they are stuck.

Other Ways to Scratch:

Besides my own books there are lots of other introductory books on Scratch, and you could read some of the reviews to determine the best books for your purpose.

My Scratch page (abhayjoshi.net/spark/scratch_page.htm) offers many more challenging project ideas (along with design documentation and sample implementation) that you could explore. And, once again, there are a ton of similar books that offer the same deal, from which you could pick. You could even pick projects from the MIT's Scratch website done by others.

Next Step: Robotics et al:

While you are exploring and learning the capabilities of Scratch, at some point you could also dip into a few related fields such as robotics, 3-D printing, mobile apps, and even AI. In fact, all these are applications of *computational thinking* and so the child can get a flavor of how programming skills and computational thinking get applied in interesting ways.

I would recommend that such learning adventures should be undertaken after a couple of years of pure Scratch. This may seem a bit arbitrary, but the logic behind that statement is that it is highly desirable for the child to be comfortable with basic ideas in computer science such as *algorithms*, *looping* (repetition), *conditionals*, *variables*, *random numbers*, etc. to really get the best out of these other learning endeavors.

There are several companies engaged in the business of teaching Robotics and other applications of CS (for example, https://www.lydnow.com/ in Pune) for children.

AppInventor (https://appinventor.mit.edu/) is an exciting Scratch-like block-based language that allows you to design mobile applications. It is also designed and supported by MIT.

This website in UK offers an introduction to AI and machine learning through Scratch: https://machinelearningforkids.co.uk/.

As you can see, it is not at all difficult to keep your child active and challenged during these 4 years of "Scratch" middle school when they strengthen their grasp of basic CS concepts, gather experience of building programs of various degrees of complexity, and also get exposed to numerous other application areas of CS.

After Scratch: Snap and Python!

After 4 years of Scratch and all the other cools things I mentioned above, your child is ready for a change. There is no dearth of options in terms of programming languages, but, the most popular these days is Python. But the jump from Scratch to Python can be traumatic, because Python is text-oriented, it has no graphical drag-drop interface, there is no animation, and the best you can do with it (at least initially) is some boring arithmetic or word juggling!

I have seen many students actually getting turned off to CS when they are pushed into languages like Python, C, C++, etc. because of their boring interface and the jungle of syntax errors.

But there is a solution. I propose a mixed approach. Transition your child to a language called "Snap!" (https://snap.berkeley.edu), which is a major upgrade of Scratch and is an excellent bridge between Scratch and Python. Snap, which its authors claim to be closest to a "real" programming language, contains a lot of CS features and concepts missing in Scratch, which are essential in Python programming. Plus, you can continue to do cool game and animation projects while using these advanced ideas in Snap. TEALS (a US-based NGO that promotes CS education in high schools) uses this hybrid approach in their year-long course: 6 months of Snap followed by 6 months of Python.

Look up my Snap page (http://www.abhayjoshi.net/spark/snap.htm) for a variety of resources including lots of interesting and challenging projects.

Author: Abhay B. Joshi (abjoshi@yahoo.com)
Last modified: 4 January 2021