# Coding for Kids

## K12 Online 2103

Hi and welcome to the K12 Online Conference for 2013. My name is Chris Betcher and this is Coding for Kids.

Many years ago, I went to school just around the corner from here, and when I was in grade 8 my science teacher, who had a part time job as a computer programmer at a local university, gave us a really interesting project to work on. We walked down to this set of traffic lights and analysed the possible paths through the intersection and collected data about the number of cars passing through. We took all that data back to school and learned to code it onto Fortran punch cards. He then took our punched cards into the university each night and returned the next day with reams of paper printouts showing graphs of simulated traffic flows. We analysed them to figure out if we could improve the efficiency of the traffic flow by changing the timing of the light sequence… we’d punch some more cards with our new data, he’d run them again and we’d repeat the process over and over until we perfected it.

Maybe it was the fact that it was a task based in the real world, or that we got to work with a computer (indirectly), or that we learned this bizarre skill of punching little holes in cards that actually meant something, or that we got to iterate our predictions over and over until we came up with something that worked, but it was the best thing I remember about school. I think it was about not just being able to use a machine to help us come up with a solution, but that we actually had **agency** over the machine. We were in control. We got to tell the machine what to do, and it did it.

Since becoming a teacher, I’ve always made a point of teaching my students how to program a computer. Over the years we’ve done a bunch of different languages, from BASIC to Scratch to Python, and almost without fail, they get the same sort of buzz from telling the machine what to do and having it do it.

Knowing how to code is something that every student should learn to do, or at least be exposed to. Not every student will grow up to become a programmer, but having some understanding of code, and more importantly having some understanding of the ideas of the algorithms and computational thinking behind it, is something that is incredibly valuable.

So in this presentation I wanted to take you through some options for exposing your kids to coding. There are lots of options out there, suitable for all ages, and if you just pick up on one of them and give your kids a chance to try it, you never know what doorways it will open for them.

## Starting Young

Let’s start with the young students. One of the key ideas of programming is sequencing, or putting instructions in a specific order. Using little programmable robots called **BeeBots** for example, very young students can pretty easily program them with a sequence of moves and turns to get through a maze or grid. This simple idea of sequencing a set of instructions is fundamental to coding.

Although I really like the idea of programming real physical objects like robots, if you have access to iPads, there are a couple of apps that can help your kids learn these same core ideas of sequencing.

**BeeBots** ([http://www.terrapinlogo.com](http://www.terrapinlogo.com/bee-botmain.php)) is also available as an iPad app, and just like the actual toy robots, it requires younger students to carefully think through the steps for getting through a series of increasingly complicated mazes. The higher levels get quite challenging!

Similar to this idea is **Daisy the Dinosaur** ([http://www.daisythedinosaur.com](http://www.daisythedinosaur.com/)), a simple programmable game that challenges students to sequence the movements of a cartoon dinosaur character to step, jump and spin, turning the sequence of algorithms into a game.

Taking this idea a step further is **Kodable** (<http://www.surfscore.com/>) which presents students with a similar series of challenges to sequence their way through maze-like pathways. However, Kodable introduces additional coding concepts, such as branching and looping. These ideas of if-then decision making and iterative repetition are fundamental to creating code. Once students grasp these concepts, it paves the way for developing more complex programming.

## Next Steps

Once students grasp these key ideas - sequencing, branching and repetition - it opens the door to a number of other interesting coding tools for young minds. The best known of these is probably **Scratch** (<http://scratch.mit.edu>), an amazing visual programming environment for kids developed at MIT. With Scratch, students can snap together simple coloured blocks to create algorithms, and with a bit of experience and thought, can actually express some quite complex ideas in code. I’m a big fan of Scratch myself, and I believe it’s something that all young students should get a chance to delve into.

This idea of assembling visual blocks of code is used in a number of other programming environments too. **Tynker** ([http://www.tynker.com](http://www.tynker.com/)) uses the same kind of idea of blocks that snap together, and although it isn’t free like Scratch, it offers an interesting built-in learning curriculum and a unique set of tools for monitoring student progress.

For iPad users, it’s worth taking a look at **Hopscotch** ([http://www.gethopscotch.com](http://www.gethopscotch.com/)). It also uses visual code blocks that can be assembled together to forms sequences of instructions, and although it is has a limited range of blocks compared to something like Scratch, the fact that it makes use of the touch screen and the accelerometer on the iPad opens up some interesting possibilities for creative programming ideas.

**Blockly** (<https://code.google.com/p/blockly/>) is a Google Code project that uses similar visual bocks of code that snap together to solve a number of coding challenges. You can load, play and hack a number of pre-made projects in Blockly so it’s great for deconstructing code, and Blockly actually forms the basis of Android App Inventor, but more on that later.

**StarLogo** (<http://education.mit.edu/projects/starlogo-tng>) takes the concept of turtles from the original Logo programming langauge develeoped by Seymour Papert and supercharges it with a whole lots of options for complex programming. Again, it uses the idea of snap-together blocks to develop flow and logic in the code, and starts to integrate some quite complex mathematical thinking.

## Moving Up

As you look at all these different tools you start to realise that the ideas behind coding are not that difficult. Sequencing, branching and looping form the basis of so many problems that can be broken into algorithms. Of course, it can get much more complex, but you’d be amazed at just how much can be achieved with these few simple ideas and a bit of maths and logic. Coding is all about solving problems, breaking them down into small steps and designing step by step solutions to deal with them.

Once you start to develop the skills and understanding of coding, you might want to get your kids looking at tools like **Alice** ([http://www.alice.org](http://www.alice.org/)), which lets you create instruction sets to control 3D models. Alice can be daunting at first, but really, it works on the same principles and ideas of most of the simpler coding tools we’ve already mentioned

**Kodu** (<http://fuse.microsoft.com/projects/kodu>), from Microsoft, is another 3D animation tool, kind of similar to Alice in the way it lets you put coded instructions behind 3D characters, and is useful for developing storytelling and animation projects.

**Game Salad** ([http://gamesalad.com](http://gamesalad.com/)) provides an environment for building game-like programs that can be easily turned into apps for iPhones or iPads. It introduces concepts of object oriented programming, which form the basis of more serious programming languages.

If you have access to Android devices you should check out **Android App Inventor** ([http://appinventor.mit.edu](http://appinventor.mit.edu/)). Based on the same code blocks used by Blockly, it includes a module for defining the buttons, forms and user interface for the device’s screen. Once the screen elements are laid out, they are then “wired up” with functions to make then do things. Students can build simple Android apps very quickly, and once they get the hang of how it works, it’s possible to develop some very interesting app ideas that can be coded up and made available in the Google Play Store for Android apps, all at no cost. Some kids will find that very motivating!

If you have the hardware to go with it, **Lego Mindstorms Robolab** ([http://www.legoengineering.com/program/robolab](http://www.legoengineering.com/program/robolab/)) let’s you program robots made with Lego. These robots, based on Lego’s NXT or EV3 units are great fun to build and program, and there are tons of resources and tutorials online, as well as a thriving Robocup culture of competitions and challenges all over the world. Robotics seems to really motivate kids, I think because it forms this really tangible bond between the coded instructions and an actual physical object that responds and reacts to its environment.

All of these environments we’ve mentioned have one very important feature in common. Not only do they allow you to create code that gives a predictable result, but they also let you see HOW that code was created. If you see someone else’s program, in Scratch for example, you can not only run the program to see its output, but more importantly you can get inside and see exactly how it was put together. In the programming world this idea is known as Open Source, and the ability to see other’s code, learn from it, copy it remix it, share it, forms a fundamental part of the culture of programming. In coding, you learn from each other. This social aspect of coding is really important.

## Getting serious

So far, most of these coding tools have not required students to actually “write” code. Because they let kids just drag blocks and arrange them into sequences, it’s possible to learn an awful lot about coding without actually needing to develop a deep understanding of programming structure or syntax.

However, once your kids grasp the core ideas of coding, many of them will want to go further and learn a “real” programming language. There are literally hundreds of programming languages, but the best ones for anybody wanting to learn more would probably be **Python** (<http://www.python.org>) or **Ruby** (<http://www.ruby-lang.org/>) Both these languages are free and can be installed onto personal or classroom computers, and there are lots of resources online for learning to use them.

In the classroom, it’s worth taking a look at **Ruby for Kids** (<http://ruby4kids.com/ruby4kids>) which will step you through the basics of programming with Ruby. If you decide to learn Ruby then take a look at Hackety Hack ([http://hackety.com](http://hackety.com/)), an online interactive tutorial to help you get up and running fast with Ruby.

If you prefer Python but have kids with iPads, you can even get **Python for iOS** ([http://pythonforios.com](http://pythonforios.com/)). It’s available in a number of versions, so whether you want to learn use the older Python 2.7 or move up the latest 3.x versions, you can get either. It’s pretty cool that you can do serious programming on an iPad! Because Python is incredibly popular as a great first language to learn, there are tons of online tutorials and websites that teach you how to get started. Check out PyGame (<http://pygame.org>) for a fun way to get started.

If you really want to do some serious coding on an iPad, then you could look at Codea ([http://twolivesleft.com/Codea](http://twolivesleft.com/Codea/)), an absolutely amazing coding environment that runs the Lua (<http://www.lua.org>) programming language from South America. Lua is yet another possible language your kids could learn, and it seems to have clean syntax and would be similar to learning Python or Ruby. Codea has powerful code completion and syntax correction built into it, and lets you build some amazing games… check out the included programs to get a taste of what’s possible, or try playing Cargo Bot from the Apple App Store, a game built entirely in Codea. And if you want to see how Cargo Bot was created, the open source code is available inside Codea. And amazingly, all of this on an iPad.

And if those languages are not enough for you, there is always Dart ([https://www.dartlang.org](https://www.dartlang.org/)), a programming environment created by Google. It’s powerful, flexible and relatively easy to learn. It produces output in Javascript, with plenty of built-in libraries and APIs that take full advantage of the web.

For your students who really want to geek out on this stuff, there are plenty of great choices. Of course, this level of coding might not be for everyone, but it is certainly for some. There are kids who absolutely come alive with this stuff, who are incredibly passionate about it. As teachers, it’s good to at least know this stuff exists so that we can connect our kids with these tools.

Happy coding!

**# Hardware**

1. Picoboards - <http://www.picocricket.com/picoboard.html>
2. Arduino - [http://www.arduino.cc](http://www.arduino.cc/)
3. Makey Makey - [http://www.makeymakey.com](http://www.makeymakey.com/)

**#Learning**

1. Grok Learning - <http://www.groklearning.com/challenge>
2. Codecademy - [www.codeacademy.com](http://www.codeacademy.com)
3. Code School - <http://www.codeschool.com/>
4. W3 schools - [http://www.w3schools.com](http://www.w3schools.com/)
5. Open Classroom - <http://openclassroom.stanford.edu/MainFolder/HomePage.php>
6. Udemy - <https://www.udemy.com/>

**#Groups**

1. Girls who code - [http://www.girlswhocode.com](http://www.girlswhocode.com/)
2. Robogals - [http://www.robogals.org](http://www.robogals.org/)
3. Tech Kids Unlimited - [http://www.techkidsunlimited.org](http://www.techkidsunlimited.org/)
4. Game Star Mechanic - [http://gamestarmechanic.com](http://gamestarmechanic.com/)
5. Home School Programming - [http://www.homeschoolprogramming.com](http://www.homeschoolprogramming.com/)

<http://www.symbaloo.com/mix/coding-coding-coding>