Proceedings of the Interdisciplinary STEM Teaching and Learning Conference

Volume 1 Article 9

5-2017

Learning to Program in Python – by Teaching It!

Bryan J. Fagan M. Ed.
Lumpkin County Middle School, bryan.fagan@lumpkinschools.com

Bryson Payne
University of North Georgia, bryson.payne@ung.edu

Learning to Program in Python – by Teaching It!

Abstract
The US Bureau of Labor Statistics predicts over 8 million job openings in IT and computing, including 1 million cybersecurity postings, over the current five-year period. This paper presents lessons learned in preparing middle-school students in rural Georgia for future careers in computer science/ IT by teaching computer programming in the free, open-source programming language Python using Turtle graphics, and discusses exercises and activities with low-cost drones, bots, and 3D printers to get students interested and keep them engaged in coding. Described herein is one pair of instructors' (one middle-school, one university) multi-year, multi-stage approach to providing engineering and technology courses, including: how to code Turtle graphics in Python; how to engage children by using short, interactive, visual programs for every age level; building cross-curricular bridges toward technology careers using 3D printing, robotics, and low-cost drones; and, how to build more advanced programming skills in Python.

Learn to Program in Python - by Teaching It!

Bryan J. Fagan, Lumpkin County Middle School
Bryson R. Payne, University of North Georgia

Abstract: The US Bureau of Labor Statistics predicts over 8 million job openings in IT and computing, including 1 million cybersecurity postings, over the current five-year period. This paper presents lessons learned in preparing middle-school students in rural Georgia for future careers in computer science/IT by teaching computer programming in the free, open-source programming language Python using Turtle graphics, and discusses exercises and activities with low-cost drones, bots, and 3D printers to get students interested and keep them engaged in coding. Described herein is one pair of instructors' (one middle-school, one university) multi-year, multi-stage approach to providing engineering and technology courses, including: how to code Turtle graphics in Python; how to engage children by using short, interactive, visual programs for every age level; building cross-curricular bridges toward technology careers using 3D printing, robotics, and low-cost drones; and, how to build more advanced programming skills in Python.

Introduction

The initial inspiration for an Engineering and Technology course at a rural middle school, which now includes computer programming, originated from the desire to provide a unique approach to teaching problem solving skills to my students. My personal observation at that time, after a decade of teaching, was that my students were overly focused on getting a correct answer and not on the process of finding solutions. With an interest in computers and a minimal background in computer programming, I proposed to my school's administration, and eventually got approved, to teach robotics during my planning period. Within a school year I had acquired, on a very limited budget, some LEGO Mindstorm Robotics kits and started preparing engineering and programming challenges for my "lego kids", as they were called by my colleagues.

The primary goal of the robotics program was for students to shift from an answer-driven attitude of learning to embracing multiple approaches and possible solutions for any given problem or challenge. At first, my students were slow to embrace the paradigm shift to problem solving and were often frustrated when it came to solving multifaceted problems with as many possible solutions.

I expected this change to be challenging for my students but did not expect it to also be a challenge to the parents who vocally expressed concern about their child's progress in the class and what they could do to better prepare them for the challenges. It took time, but by the end of the class my students were asking good questions, seeking multiple solutions, modifying their approach when necessary, and collaborating with each other.

The robotics program lasted for several years and was, in many ways, a successful attempt at robotics, programming, and changing the way my students approached problem solving. The robotics program showed my students were highly motivated, problem solvers, when challenged and would be interested in a multi-grade connections course in engineering and technology. It also proved to be the foundation of a larger engineering and technology program, thanks to the ability to demonstrate significant student demand through high participation rates in both robotics classes and after-school programs.

Background

Much, if not the majority, of the literature on middle-grades computing curriculum concerns the use of visual applications, like Scratch, Alice, or even Flash (described below), to teach introductory programming, and many school systems start their programming courses as special electives or after-school programs. Webb and Rosson (2013), in one typical example, used the drag-and-drop, block-based programming environment Scratch to teach an outreach enrichment program for middle-school girls. The researchers used scaffolded activities, stepping from building a story, to solving a maze, to storing data in a list, to working with sensors and motors.

Before Scratch, previous researchers had even employed visual tools like Macromedia (now Adobe) Flash, the once-popular Web animation and programming tool, to teach computing concepts in middle school using animations and simple 2D games. One such team developed an after-school program focused on game programming in Flash (Werner, Campe and Denner 2005), and found that IT fluency overall improved for middle-school girls who created Flash games.

LEGO robotics have also been popular tools in teaching introductory programming concepts and in stimulating STEAM interest and motivation in middle-schoolers (Kaloti-Hallak, Armoni, and Ben-Ari 2015). Like Scratch, the LEGO programming software allows easy drag-and-drop blocks to form the logic

https://digitalcommons.georgiasouthern.edu/stem_proceedings/vol1/iss1/9

DOI: 10.20429/stem.2017.010109

of a program, but the effects can be seen in real life by running the "code" on a LEGO Mindstorms robot. In our program, we had access to a limited number of LEGO robots, and we wanted to take advantage of both the variety of activities and the inherently interesting "hook" of getting students to program the bots to perform tasks in the live classroom environment. However, robots alone could not fill a full nine-week course at the scale we had the opportunity to teach programming, let alone a full semester or eventually a year of coding.

Other researchers have used different visual software, like the AgentSheets platform used in Scalable Game Design (Bennett, Koh, and Repenning 2011) that allowed students to build visual games like Frogger relatively easily. Others used virtual 3D software like Curiosity Grid (Hulsey, Pence and Hodges 2014) in a one-week summer coding camp environment for girls, to motivate greater STEAM interest in middle-school females. Still others have developed entire CS Principles curricula using game-based systems like ENGAGE (Buffum et al. 2014).

But, similar to Scratch and other visual programming tools, software packages like these were built specifically for teaching, not for programming. This introduces two significant obstacles in developing coding fluency and problem-solving ability in programming in general. First, the software limits the extent of the programming students can do; by building teaching-based tools, some essential low-level programming constructs are unfortunately left out, forcing students to feel like they're not doing "real" programming. Second, there is often a learning curve in figuring out how to use an already-limited tool, taking time away that could have been spent learning how to solve problems by programming in a text-based language.

More recently, partly in response to these issues, there has been a trend toward using text-based programming languages, most notably, Python. Armony, Meerbaum-Salant, and Ben-Ari (2014) studied middle school students who had studied Scratch versus those who had no programming experience at all, and found that, while the Scratch users could pick up concepts faster in a text-based programming course in high school, there was *no significant difference* in overall achievement at the end of the high-school class. This seemed to indicate that there was an initial benefit to learning the concepts taught in Scratch, but that the benefit faded over time and had less lasting impact on "real" programming ability in text-based languages by the end of a second course.

Tabet et al. (2016) designed a middle-school curriculum that started

out in Alice, a 3D drag-and-drop environment used to create animated scenes, for learning basic programming concepts, but quickly progressed to Python to convey more advanced problem-solving skills in a text-based language. The authors were attempting to achieve a better "mediated transfer" of concepts between the visual Alice tool and the text-based Python language, and found some positive impact on performance for students who learned Alice in seventh grade followed by Python in eighth grade. However, this was in a middle school that provided two full years of programming instruction with support from four university faculty.

Implementation

I knew I would have to be resourceful, as is the case for teachers in many smaller community schools, to begin an engineering program in rural north Georgia. The robotics program was successful, but it was not a cost-effective platform for teaching a multi-grade connections course in a school system that currently did not have a budget for engineering and technology. While reading through the Georgia Performance Standards (GPS) for Engineering and Technology and preparing the course curriculum, I began actively looking for places to integrate computer programming as a cost-effective means to teach the engineering and technology subject-matter content. In the first iteration of a "real" technology course at my school, I found myself teaching up to twenty-six students per class, six classes per day, in nine-week rotations, for a total of twenty-four different classes in one school year. Even with the drastic price drop for engineering and technology resources (such as Arduino, Raspberry Pi, and 3D printers), I knew I would have to use coding to teach the GPS standards, as well as career-relevant skills, and keep the cost of the class per student as low as possible. My school had some refurbished computers that were not being used, and an almost closet-sized classroom that I could use as a makeshift engineering room and computer lab. I was eager to get started, so overlooking some obvious challenges was easy from the start, but they would have to be addressed as the school year, and the development of the program, progressed.

I chose to use a "real" text-based programming language, Python, from the start, with very visual programs based on Turtle graphics to give students immediate, graphics-based feedback as they developed basic through advanced programming skills. My primary resource for teaching computer programming was a coding book by my co-author for this paper, titled T*each Your Kids to*

https://digitalcommons.georgiasouthern.edu/stem_proceedings/vol1/iss1/9
DOI: 10.20429/stem.2017.010109

Code. I divided the book into two parts so it could be spread out over two years (sixth and seventh grade). This afforded me time during the short quarter (nine weeks) to teach engineering and computer programming content without having to sacrifice time in either area. I would also have just enough time to properly introduce coding to a student body that had zero programming experience.

The first five chapters of *Teach Your Kids to Code* were what I taught to my sixth graders, introducing the concepts of basic coding, loops, conditions and variables with colorful, visual apps in Turtle graphics. The coding unit lasted four weeks and amazingly my sixth grade students used the Python programming language to write around 40 different programs in that time. The seventh grade students used the second half of the coding book to get more involved in functions, timing, animations and game programming. This was the continuation from the introductory chapters and included more advanced programming concepts.

Because the material is brand new to the student body, the course is being phased in over a three-year period, and we're in the second year. Next year the eighth grade students will be learning Java, which is a more abstract but even more widely used programming language and a more functional skill set for someone interested in computer programming for AP Computer Science and college classes. Furthermore, I decided to enhance the programming curriculum through creative use of LEGO robots, programmable quad-copter Parrot Mini Drones, and 3D printers, both to engage students in more physical, kinesthetic activities while coding, and to interest students in broader STEAM applications and technologies across the curriculum beyond mere programming.

One major challenge that needed my attention was the classroom space. My classroom was way too small for teaching a course that included the need for computer equipment, engineering equipment, storage, materials and peripherals. An ideal environment for an engineering course, that included computer programming like my course, realistically required each student to have his or her own computer and sufficient space to interact with engineering equipment, such as drones, 3D printers, and electronic devices safely. I could not change the size of the room, so arranging the space as creatively and efficiently as possible was my only option. That meant only having twelve computers and six tables for twenty-six students. Initially I had two to three students writing one program at a time on one computer. This led immediately to disruptive behavior and a lack of inclusive learning, as the student holding the book was not

learning at the same pace, if at all, as the student writing the code. This challenge needed to be resolved quickly and, for me, the solution was to use the school's computer labs whenever possible. Fortunately for my situation, my school has a separate computer lab for each grade that I could usually schedule several times per week when needed. This may not be an option for every school, while others may have computer labs in almost every classroom, but my recommendation is to get the computer-to-student ratio as near to 1:1 as possible. I rotated, three times a day, between the different computer labs using a mobile cart to hold the coding books, but having one computer per student while programming reduced disruptions (almost completely) and greatly increased student interest and confidence in coding. The students were more excited each day we visited the computer lab, took more ownership of their programming, and were better able to correct errors and write programs in Python.

There are plenty of off-the-shelf kits for engineering and coding that are affordable and easy to integrate into the classroom. One option is a Parrot Mini drone that allows the user to fly manually, or by using block programming through an application called Tickle (as of now only available through the iTunes store) or Tynker, available for both Android and iOS devices. These drones are affordable ($59-75 or so on Amazon), so several can be purchased for group projects, and they are extremely durable. My students were able to use what they had learned about programming to code flight plans directly into Tickle then watch as their drone took off, flew around the gym, performed tricks, then landed safely.

Another two options that are an excellent mix of engineering and programming are Arduino and Raspberry Pi. Both of these electronic sets are extremely affordable, easy to set up, modify, and program. The Internet has plenty of great projects for both and most provide step-by-step instructions and downloadable programs to run. The options on the market right now are limitless, but not all STEAM products are created equal, so be sure to research what you plan to buy before spending significant money. Consider getting one or two devices as a mini-pilot, especially if you have a few highly motivated and capable students that could attempt a few labs and projects, then make a presentation to the class (or to your administration, asking for funding for full classroom sets).

https://digitalcommons.georgiasouthern.edu/stem_proceedings/vol1/iss1/9
DOI: 10.20429/stem.2017.010109

Getting started in STEAM
• Look for free resources, and start with what you have. You can easily begin coding if your school has an existing computer lab. If not, use one computer in your room as a lab station, and rotate students on and off.
• Keep costs low by purchasing affordable kits in small quantities, and let groups of students create projects together as teams.
• Get parents involved by hosting STEAM nights that include student-led presentations and computer programming.
• Demonstrate to your administration the effectiveness of STEAM teaching by inviting them to program with the students, observe student created projects, and attend a STEAM night.
• Use student achievement and student/parent interest to justify a larger budget for STEAM related resources and even an Engineering and Technology course.

Conclusions

The results of the Engineering and Technology class have been evidenced in several areas since I began teaching the course last year. I have seen a definite increase in student interest, both male and female, for engineering, technology, and computer programming-related topics. My students have shown a marked increase in the desire to pursue an engineering or programming related career, and they often inquire about what courses our local high school offers in engineering, technology, and computer programming. My students have also demonstrated greater ability and interest in peer collaboration, shared problem solving, and they are far more comfortable learning without having a clear, definable answer to challenges.

I have also learned over the last year that an engineering program is essential if we, as educators, want to best prepare our students for the workplace they will be entering—a workplace in desperate need of persons knowledgeable of and comfortable with engineering, technology, and computer programing. I also learned that, as part of that program, computer programming in some degree must be included. Many of the electronic components and products we used in the course to learn engineering, including 3D printers, drones, robotics, and Arduinos, were all modifiable using computer programming. My students were quick in insisting we find ways to modify, reprogram, or "hack" everything in the classroom. I agreed, and we quickly set about, over several weeks, teaching ourselves the same standards I had planned for in my "official" lesson plans.

Last, but not least, I have learned that teaching engineering, technology,

and computer programming can be frightening if you have little to no experience, but it can be done, and is honestly easier than I first imagined. One thing to remember is that you do not have to know everything about every programming language, hardware platform, type of technology, or electronic device. I discovered that most students are very eager to learn independently and then share what they have learned with their peers. Allowing them to do this worked so well that I took every Friday off from teaching so they could work on independent technology projects. I used the Georgia Educational Technology Fair categories (*http://www.gatechfair.org/categories*) as a blueprint for their projects, grading rubrics, and instructions. At the end of the course they took great pride in presenting to the class their projects and what they had learned.

As for coding, starting with a programming language that is easier to read and write, such as Python, will help grow your confidence and will definitely be easier for students new to programming to learn. Writing the program for yourself prior to teaching is important for targeting potential pitfalls, identifying common errors, and areas where students will be able to be creative and add their own code. But, once the students jump in, letting them explore and try new things, and being able to respond to questions with, "I'm not sure, let's try and see what happens!", has been easier, and more fulfilling and instructive to me, personally, than any other experience in my teaching career.

As educators, by nature, we are resourceful and inquisitive. Teaching and learning computer programming in many ways requires the same skill set. Going online and searching for solutions to programming errors or problems is very helpful. Being inquisitive and challenging yourself to write good programs will only encourage your students to exhibit the same behavior. Add to that an honest dose of being willing to try, fail, and figure out mistakes to build something new, and you can teach yourself to code, while teaching it to your students.

https://digitalcommons.georgiasouthern.edu/stem_proceedings/vol1/iss1/9
DOI: 10.20429/stem.2017.010109

References

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From Scratch to "Real" Programming. *Trans. Comput. Educ.*, 14(4), 1-15. doi:10.1145/2677087

Bennett, V. E., Koh, K. H., & Repenning, A. (2011). *CS education re-kindles cre ativity in public schools*. Paper presented at the Proceedings of the 16th annual joint conference on Innovation and technology in computer science education, Darmstadt, Germany.

Buffum, P. S., Martinez-Arocho, A. G., Frankosky, M. H., Rodriguez, F. J., Wiebe, E. N., & Boyer, K. E. (2014). *CS principles goes to middle school: learn ing how to teach "Big Data"*. Paper presented at the Proceedings of the 45th ACM technical symposium on Computer science educa tion, Atlanta, Georgia, USA.

Hulsey, C., Pence, T. B., & Hodges, L. F. (2014). *Camp CyberGirls: using a virtual world to introduce computing concepts to middle school girls*. Paper presented at the Proceedings of the 45th ACM technical symposium on Computer science education, Atlanta, Georgia, USA.

Kaloti-Hallak, F., Armoni, M., & Ben-Ari, M. (2015). *Students' Attitudes and Motivation During Robotics Activities*. Paper presented at the Proceedings of the Workshop in Primary and Secondary Computing Education, London, United Kingdom.

Tabet, N., Gedawy, H., Alshikhabobakr, H., & Razak, S. (2016). *From Alice to Py thon. Introducing Text-based Programming in Middle Schools*. Paper presented at the Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, Arequipa, Peru.

Webb, H., & Rosson, M. B. (2013). *Using scaffolded examples to teach compu tational thinking concepts*. Paper presented at the Proceeding of the 44th ACM technical symposium on Computer science education, Denver, Colorado, USA.

Werner, L. L., Campe, S., & Denner, J. (2005). *Middle school girls + games pro gramming = information technology fluency*. Paper presented at the Proceedings of the 6th conference on Informa tion technology education, Newark, NJ, USA.