A STEM-based, Project-driven, Introductory Programming Class for Pre-service Teachers

Prof. Wesley G. Lawson, University of Maryland, College Park

Prof. Lawson has earned five degrees from the University of Maryland, including a Ph,D, in Electrical Engineering in 1985. In his professional career at College Park, where he has been a full professor since 1997, he has worked on high-power microwave devices, medical devices, and engineering and STEM education. He is an author or coauthor on 5 books and over 70 refereed journal articles and 200 conference presentations and publications.

Dr. Jennifer Lee Kouo, Towson University

Jennifer L. Kouo, is an Assistant Professor in the Department of Special Education at Towson University in Maryland. Dr. Kouo received her PhD in Special Education with an emphasis in severe disabilities and autism spectrum disorder (ASD) from the University of Maryland at College Park. She is passionate about both instructional and assistive technology, as well as Universal Design for Learning (UDL), and utilizing inclusive practices to support all students. Dr. Kouo is currently engaged in multiple research projects that involve multidisciplinary collaborations in the field of engineering, medicine, and education, as well as research on teacher preparation and the conducting of evidence-based interventions in school environments.

Ms. Vaishnavi Murthy, University of Maryland, College Park

Vaishnavi is a senior undergraduate student at the University of Maryland, pursuing a degree in Electrical Engineering.

A STEM-based, project-driven, introductory programming class for pre-service teachers (Work in Progress)

Abstract – We detail an introductory programming course developed and delivered to pre-service teachers and motivated by the need for a larger STEM+C workforce. The course had two 50-minute lectures per week that introduced Python structure, syntax and keywords, standard and custom modules, best programming practices, etc. PowerPoint lectures were interspersed with active learning assignments. There was also one three-hour laboratory every week that began with a brief presentation of the engineering topics that were relevant for that week. All labs were performed in groups and followed the project-driven learning (PDL) approach. The details of this novel course are described, and the results from the first offering of the course are presented. Survey results from forty-two students from the College of Education regarding the possible value and likelihood of taking a PDL programming course will also be summarized.

Infusing computational thinking skills in K-12 education is essential for advancing the teaching and learning of Science, Technology, Engineering, Mathematics and Computer Science (STEM) in the 21st century and ensuring the competitiveness of the United States in the global economy [1]. To date, there have been a number of emerging efforts to integrate computational thinking with STEM education [2], and there are many opportunities for students to learn about computers, computer programming and computational thinking in K-12. For example, Carnegie-Mellon University, Purdue University, the Computer Science Teachers' Association (CSTA), the International Society for Technology in Education (ISTE) and others are leading the way to bring computer science and computational thinking to K-12 through programs like CS4HS [3,4]. Also, in elementary and middle schools, a student can take enrichment programs in Scratch [5], Minecraft programming, LEGO, and GameMaker programming [6].

However, opportunities for pre-service and in-service teachers to learn computational skills [7, 8], have not kept pace. There is an urgent need for K-12 teachers to develop the pedagogical skills necessary to enhance student learning of computational thinking skills in STEM contexts [9]. There is also a general need (1) for teachers to understand technology use in the context of the STEM application, e.g. engineering design, (2) for redesign of pre-service courses and in-service workshops aimed at integrated STEM instruction, and (3) for STEM teachers of different content areas to work together [10].

There has been considerable work in the STEM community, in particular in Computer Science and Engineering, to bring the PDL approach to introductory classes with programming learning objectives. Generally, the PDL computer science classes focus on software applications [11,12]. Engineering PDL classes entail hardware applications, but often focus on introductory engineering design [13-16]. In such freshman design classes, programming is usually taught in a 2-3 week module and while all students may be assessed on some basic programming knowledge, skills and abilities (KSAs) [17], projects typically involve large groups where only a few develop significant programming KSAs.

There have been several efforts to bring computational thinking and computer programming KSAs to pre-service and in-service teachers [18-20]. Those efforts typically involve software-only applications in a language like Java or in a visual programming language like Scratch.

For the past several years, we have offered a novel introductory C programming course to electrical engineering students at the University of Maryland [21-23]. This course included partner-based programming assignments emphasizing computer-controlled hardware-driven projects and a final multi-week group project utilizing Raspberry Pi (RPi) computers. This project looked at students' self-efficacy beliefs and outcome expectations as compared to students who took a traditional programming course and the PDL students left their course with a superior self-image regarding their fitness as engineers and an improved understanding of the role of computer programming in their profession. [24]

Our introductory Python programming course is designed to introduce programming concepts and computational thinking with STEM applications to preservice teachers with little or no previous programming experience, and is modeled after our previous C course, albeit with changes to the programming language, the anticipated KSAs of the students entering the course, and modified expectations for learning objectives. Our Python course introduces students to Python 3 programming in the context of simple STEM applications and also teaches elements of the engineering design process and hardware (sensors and actuators) that are needed to design, build, and run STEM experiments and applications. The topics are presented in a just-in-time format as students work towards the principle course objectives to learn how to effectively integrate computers and associated hardware into the exploration of STEM projects.

In addition to the content instruction, students participate in an authentic design experience in which teams of students put themselves in the shoes of engineers who were contracted to design and implement a computer-based system for a STEM application. Teams have to develop the project specifications in conjunction with project stake-holders, which in this case are in-service teachers looking to augment their STEM instruction. Based on stakeholder input, the teams must produce a computer-controlled or computer-enhanced project and/or experiment to explore more deeply the content of the STEM topic. Students have to design, realize (develop code and system), and analyze their project. Teams must debug their codes and troubleshoot their hardware. They also must evaluate their system's performance in terms of data errors and outliers, input errors and other unanticipated usage and results, and redesign their projects to increase robustness.

Course objectives for the Python programming class are listed in Table I. Student assessment to gauge the relative achievement of course objectives was done through regular homework assignments (outside of lab), occasional quizzes, lab reports, a final project report, and a midterm and final exam. Typical homework assignments could be done in computer labs or on personal computers (Python is available free on most platforms). Example homework assignments would be to solve quadratic equations, do linear and exponential regression, plot data on linear, semi-log and log plots, represent data with pie charts, roll dice in games and determine the winner based on game rules (e.g. for the game *Risk*), perform temperature and other unit conversions and generate or interpret Morse Code. Exams and quizzes would be closed book and would require writing code, modifying or interpreting code, and correcting logical and syntax errors in code. Lab reports needed to be sufficient for other class members to duplicate their results.

Students utilized Raspberry Pi (RPi) computers in the laboratory section and there were eight labs that were completed during the semester. Many of the Python lab projects had the same specifications and utilized the same hardware as the C programming labs. Students were also issued an RPi that they could use outside the lab to complete assignments. For the first lab, students assembled the RPi computers from a kit and learned how to generate and run Python codes. They also learned to use the hardware interface to vary the brightness of light emitting diodes (LEDs). Subsequent labs required students to turn lights on and off based on the ambient light, explore the use of temperature and distance sensors, use different communication protocols to control magnetic and distance sensors, use motion detectors to automatically record video, and use accelerometers to infer the velocity and location of moving objects. Finally, students utilized various modules in the lab to integrate graphical user interfaces into their labs and display data graphically in real time. Students selected as a final project to develop a "cloud in the bottle" experiment in which they pressurized a bottle that contained some water and then measured the pressure and the reflectivity of the "cloud" that developed once the pressure was rapidly reduced.

Table I. The stated objectives of the Python Programming class for pre-service teachers.

An appreciation for the enabling role of computers and computational thinking in STEM applications.
Operational familiarity with elementary Python programming concepts: program control flow, basic and collection data types, custom and off-the-shelf modules, logic and arithmetic operations, methods and functions.
The ability to utilize good programming practices to write efficient, clear, and maintainable code.
The ability to debug, load and run code to solve STEM problems and demonstrate STEM concepts.
An understanding of basic circuit theory and the operation of basic electronic components, sensors and actuators.
The ability to work effectively in teams.
The ability to communicate effectively in written and oral formats.
The ability to use elements of computational thinking, engineering design, and programming & hardware knowledge, skills and abilities to develop computer-controlled systems that can enhance existing STEM investigations.

For the first offering, only two students took the course, as it was mis-advertised as a course only for future middle school STEM teachers. Both students were female and both achieved an "A" grade in the course based on the technical assessments and expectations. One student was a freshman and the other student was a sophomore; both intend to teach middle school math or science, looking for certification in grades 4-9.

At the beginning of the semester, neither student felt confident about their abilities to learn how to program or how programming could impact their careers. After the final class, the two students participated in a focus group to learn more about their experience in the course. In line with Social Cognitive Career Theory (SCCT) [24], the students were asked about their self-efficacy beliefs and outcome expectations. When asked how confident they felt about incorporating what they learned into their professional activities, one student felt she understood the content overall, but was "unsure of the ability to explain or teach the content to students and answer their questions." whereas the other student felt "confident about teaching the content presented at this level." The first student felt stronger about being able to teach plotting and data charts and thought that maybe she could lead a student programming club.

Both students were unsure how they could incorporate programming in their curriculum, with one student stating she would "need to be creative to apply coding in math or science within a more structured curriculum." Both thought that starting a programming club would be possible.

When asked how programming relates to their intended profession, the freshman remarked that it allowed a "greater integration of technology that has not been experienced before in her own education," and the sophomore considered "potentially having a Coding Corner in the classroom" and she felt it "opened up a new door of opportunities and hobbies."

When asked what aspect of the programming class was most productive for them, the sophomore replied that the labs and the concrete examples showing in the impact on hardware were most productive, while the freshman felt the "graphs near the end (of the semester) which helped to connect everything in the course together" were most productive. In both cases, it was visual content that helped enhance their confidence and enthusiasm for coding.

When asked what changes they would recommend for the class, the freshman felt the instructors should "consider the usage of middle-school level mathematics as opposed to more advanced mathematics to help with application," and that there should be other adjustments based on the pre-service teachers typical "foundational knowledge or prior knowledge." The sophomore agreed and thought there should be a pre-assessment given to the students "to appropriately prepare course sequencing, activities, and instruction," as well as a closer interaction / alignment with local school systems' curriculum. The main additional recommendation was to make the course available "to students specializing in math and science across all education majors" and tailoring the coding assignments "from simplistic to intricate based on age/grade levels."

The two students were asked to reflect at the end of their course on their main takeaways from the course. The freshman student stated: "*Although we may be ending our course work with programming here, we have a newfound interest in the subject that we will bring with us the rest of our academic careers and beyond!"* and the sophomore replied: "*We both concluded that although we may be more focused on the S and M in STEM, we now want to incorporate our knowledge into the classroom. By having a 'Tech Corner' of sorts and having programming equipment available to our students, we believe we will promote more than just what our degree says we can do."*

For our on-line survey, all undergraduate students from the College of Education were invited to participate in a survey that was developed to gain insight into pre-service teachers' attitudes toward programming classes. Forty-two students completed the survey. Twenty-three self-identified as focusing on elementary/early/special education. Fourteen identified as secondary education with only three concentrating on middle-school. Four did not identify a concentration and one listed Art Education. Four also self-identified as pre-service STEM educators. Forty females and two males responded. Thirty-five students (83% of the students) indicated that they had no previous programming experience at all. In contrast, over 90% of the freshman engineering students in the PDL C class *did* have previous programming experience.

A five-point scale was used for most questions. Three indicated a neutral response, one indicated strong disagreement and five indicated strong agreement. A sample of some of the responses are shown in Table II. Students were also asked what level of programming skill they believe they would need in their regular duties as an in-service teacher. For this question a one meant zero skill; a two meant little skill (could understand and make minor modifications to small codes); a three meant moderate skill (could write codes from scratch with dozens of lines); a four meant fluent (could debug others' moderate-length codes and write complicated codes from scratch); a five meant mastery (Could effectively teach others to program). The average response to this question was 1.93 ± 0.81. No students felt mastery was important. One person felt fluent was important, 41 (almost 98%) felt moderate or less was adequate. While most students believed that programming skills could enhance their careers, and they were mildly pessimistic on relative the level of difficulty, most would not take a programming class unless they were required to. A slightly positive response occurred for a programming course as a general education elective – counting towards a degree but not required. Perhaps the results would have been different if only pre-service STEM teachers or only secondary education majors were polled, but we chose to survey all pre-service teachers as coding is generally considered to be a useful skill to be taught throughout K-12 (consider, for example the "Hour of Code" and the wealth of coding enrichment activities that are available to students at all levels.)

Table II. Selected responses from the survey taken by 42 pre-service teachers.

Question	Average ± Deviation
I believe that having good programming skills will add value to my future intended career.	3.62 ± 1.06
I expect an introductory programming course would be quite difficult.	3.24 ± 1.01
I would be willing to take an introductory programming course only if it were required to graduate.	3.71 ± 1.09
I would be willing to take an introductory programming course if it were a general education elective.	3.31 ± 0.98
I would take an introductory programming course if it fit my schedule even if it didn't count toward graduation requirements.	2.38 ± 1.06
An introductory programming course should teach software only.	2.86 ± 0.75

We intend to teach the class again in one year, modifying it based on the feedback and data we have accumulated. In particular, we will clearly advertise to all K-12 pre-service teachers, and we will work more closely with faculty from the College of Education on project content, in addition to a pre-survey given to the students the first week of class. Based on survey feedback, we will also add a unit at the beginning of the course to illuminate the utility of computers and computational thinking in the K-12 STEM curriculum. This new module will highlight the call from government and educational institutions for the integration of computational thinking and computer programming in STEM education and will survey projects underway to help meet this need. Finally, there will also be multiple specific examples given of how computational thinking and programming can enhance the KSAs acquired in STEM lessons. There will either be a guest lecture or multiple videos from in-service teachers who can describe their own personal activities in STEM+C education. We will also work to transform this course into an on-line version suitable for in-service teachers. The freshman student has actually volunteered to be a teaching assistant for the lab part of the course.

References

[1] President's Information Technology Advisory Committee (PITAC 2005, p.1). Computational Science: Insuring America's Competitiveness. Washington, DC.

[2] (Swaid, 2015). Bringing computational thinking to STEM education. *Procedia Manufacturing*, *3*, 3657-3662.

[3] https://www.cs.cmu.edu/outreach/cs4hs-2006-2013.

[4] https://www.cs.purdue.edu/outreach/news-items/csedweek-programming-challenge-2013.html

[5] https://scratch.mit.edu/

[6] http://www.thoughtstem.com/minecraft

[7] http://blogs.edweek.org/edweek/curriculum/2016/04/arkansas_teachers_get_lessons_computer_coding.html

[8] www.Lynda.com

[9] https://thejournal.com/articles/2013/10/03/integrating-programming-with-core-curriculum.aspx

[10] https://gostem.gatech.edu/en/professional-development

[11] J. Kay, M. Barg, A. Fekete, T. Greening, O. Hollands, J. H. Kingston, and K. Crawford, "Problem-Based Learning for Foundation Computer Science Courses," Comput. Sci. Educ., vol. 10, no. 2, pp. 109–128, (2000).

[12] Andrey Soares, "Problem based learning in introduction to programming courses," J. Comput. Sci. Coll., vol. 27, no. 1, (2011).

[13] G. W. Reckenwald and D. E. Hall, "Using Arduino as a Platform for Programming, Design, and Measurement in a Freshman Engineering Course," in n *American Society for Engineering Education Annual Conference*, 2011.

[14] J. D. Steinmeyer, "Project-Based Learning with Single-Board Computers," in American Society for Engineering Education Annual Conference, 2015.

[15] X. Zhong and Y. Liang, "Raspberry Pi: An Effective Vehicle in Teaching the Internet of Things in Computer Science and Engineering," Electronics, vol. 5, no. 3, p. 56, (2016).

[16] C. Carlson and D. Day, "Transformation of an Introductory Computer Engineering Course Utilizing Microprocessors and a Focus on Hardware Limitations," in American Society for Engineering Education Annual Conference, 2017.

[17] K. M. Calabro, K. T. Kiger, W. Lawson, and G. Zhang, "New directions in freshman Engineering Design at the University of Maryland," Proc. - Front. Educ. Conf. FIE, pp. 6–11, 2008.

[18] Irene Govender, and Diane J.Grayson, "Pre-service and in-service teachers' experiences of learning to program in an object-oriented language," Computers & Education Volume 51, Issue 2 Pages 874-885, (2008).

[19] Nathan Bean, Joshua Weese, and Russell Feldhausen, "Starting from scratch: Developing a pre-service teacher training program in computational thinking," presented at 2015 IEEE Frontiers in Education Conference (FIE).

[20] Aman Yadav, Chris Stephenson, and Hai Hong, "Computational Thinking for Teacher Education," Communications of the ACM, Vol. 60 No. 4, Pages 55-62, (2017).

[21] W. Lawson, S. Secules, S. Bhattacharyya, and A. Gupta, "An Application-based Learning Approach to C Programming Concepts and Methods for Engineers," in American Society for Engineering Education Annual Conference, 2016.

[22] W. Lawson, S. Secules, A. Elby, W. Hawkins, T. Dumitras, and S. Bhattacharyya, “Traditional versus Hardware-driven Introductory Programming Courses: a Comparison of Student Identity, Efficacy and Success,” in American Society for Engineering Education Annual Conference, 2017.

[23] S. Secules and W, Lawson, “Description and Evaluation of a Novel Hardware-Based Introductory Programming Course," accepted by Advances in Engineering Education.

[24] Lent, R. W., Brown, S. D., & Hackett, G., “Toward a unifying social cognitive theory of career and academic interest, choice, and performance [Monograph]”. Journal of Vocational Behavior, 45, 79-122, (1994).