Learning Python in an Urban After-School Middle School Program

Suzanna Schmeelk[1], Alfred Aho[2], Stephanie Wortel-London[3], Kanika Bansal[4]

Abstract

This research explores the psychology of programming and the pedagogical environment in an after school urban middle-school program located in a New York City urban school. The program, sponsored by the Department of Education, is aimed at teaching under-represented urban middle-school students to learn how to write computer programs. Because many of the students had limited, if any, exposure to programming skills, the Python language was selected to introduce computing concepts. The computing concepts and the fostered pedagogical environment were implemented in one-hour after school sessions over 15 weeks in which the students were encouraged to develop computing communities while working on computational thinking concept strands. Our findings provide unique insights into intervention constraints for an urban after school program which can be used to guide and inform further after school computer learning research

Keywords: innovative teaching, learning metodologies, pedagogical environment, programming skills

1. Intervention Design

We describe in-depth the design choices of our weekly after-school interventions as informed by unique constraints and the urban environment. When we started the after-school intervention research, there was little, if any, literature informing researchers on either successful intervention design or unique environment constraints. This research paper attempts to fill the gap in literature to address issues unique to the after-school urban environment which needs to be understood in-depth and prepare for in advance of the programming intervention.

An effective pedagogical environment has been shown to influence the overall learning outcomes. For example, Ball et al. [1] showed that many mathematics curricula and classrooms emphasize the learning of algorithms and procedures rather than concepts. In such environments, as cited by Kamii and Dominck [2], students are expected to relinquish their own intuitions and disconnect content from the underlying concepts. Knowing these traditional weaknesses in educational pedagogy, we reinforced concepts through a weekly tri-instruction model: syntax lesson, semantic lesson and a software engineering lesson. Specifically, the aforementioned known pitfalls in education, lead us to create a pedagogical environment framed around individual student mean-making, student team building and open environments while taking advantage of interactive learning tools. The students were encouraged to work in groups and use personal representations to construct their final project. We describe each component of the model in the following sections.

1.1 Classroom Leadership

Even in an after-school program, classroom leadership is essential for durable learning. Classroom leadership can either transpire from the peers online, peers in the classroom or from instructors. The classroom leadership component of intervention design reinforces the research of Tenenberg et al. [3], where it was reported that bringing computing professionals into the student learning experience improved learning success. Our pedagogical environment was thus designed to be led by computing and education leaders with diverse backgrounds and experiences.

1.2 Inquiry

Student learning is not a simplex linear process; the human mind is complex. Although we faced intervention time constraints of less than one hour on a weekly basis, it has been shown that student learning guided by inquiry produces more durable learning. Inquiry is an essential component for developing student reasoning [4], [5] and therefore a successful intervention should establish an environment where student inquiry leads student learning. As such, developing open situations

[1] Department of Computer Science, Columbia University, United States
[2] Department of Computer Science, Columbia University, United States
[3] Director of Education at the New York Academy of Science, United States
[4] Department of Mathematics, University at Buffalo and US-ARL, United States

(Fosnot and Dolk [5]) and tasks (Mueller, Yankelewitz and Maher [6]), has been shown to be effective for building long-term understandings. Their research guided the instruction choices we made.

1.3 Classroom Environments

Our interventions were designed to foster a unique classroom environment where students could discuss together while learning independently. The after-school program offers unique constraints such as continual fluctuation of students in and out of the class on a weekly basis. Our interventions were constructed in such a way that group and pair-wise open discussion occurred in parallel in every session.

1.4 Interactive Tools

Our after school program created a unique context since students would miss multiple sessions over a few weeks creating a flux of different learning levels as well as different age ranges. We found that we needed a method to bridge student learning when they missed sessions without being left too far behind so that they could never catch-up in class. Therefore, after examining the available resources, we decided to include an online component of the class, based in part on research at large (e.g. [7]). The research has shown that interactive tools are successful in improving the learning outcomes of students.

1.5 Computational Thinking

Our interventions were designed to foster computational thinking by giving students mini-real world problems which they would need to be mapped down-into a computer programming language representation. Our interventions design choices were selected as it is well known in literature that building students' computational thinking is an essential component for developing durable computing skills (e.g.[8],[8]) Bringing computing skills to the K-12 level has been a research focus for many years. One of the hardest components is the effective exposure of computational thinking. Our interventions were designed around student personal interests, which the students regularly expressed with positive feedback.

1.5.1 Representations

Our interventions were designed around student interests. We formally surveyed the students before the first week of class to gain an understanding about their backgrounds and interests in computing. During this pre-survey, we decided to abstractly follow the Ginsburg's strategy for clinically-interviewing students. In our weekly interventions we would openly discuss student personal meaningful representations which would be the focus of their weekly software engineering activity or software engineering final project. Our methodology reflects the research of Fosnot and Dolk [5], Schmeelk [10] and Suthers and Hundhausen [11]. The students appeared to enjoy seeing what other students wrote as much as they enjoyed writing their own meaningful programs to take home.

1.5.2 Understanding

Our intervention design was informed by the work of Davis and Maher to help students build durable computing concepts. Davis describes *Understanding ``Understanding"*, [12], in his research when he states, ``Put in its starkest terms, this theory postulates that one gets the feeling of *understanding* when a new idea can be fitted into a larger framework of previously-assembled ideas. A metaphor that reflects this quite well is the notion that one assembles ideas in one's own mind much as one assembles a jig-saw puzzle." Davis argues against superficial verbal learning and following algorithmic recipes. Our weekly interventions were specifically selected to reinforce assimilation paradigms as published by Davis and Maher [13].

2. Session Structures

Each session was divided into five high-level components over ten units. First, we would discuss the prior learning. This open discussion encouraged students to recall learnings from the prior week as well as having students listen to other student notions. Fosnot and Dolk [7], refer to this as a *congress*. Second, we would bring in the concepts for the week by asking the students about thing they personally do. We would introduce technical language like *control flow*, *function*, *data structures*, *algorithm* and *software engineering* through terms that they use in their daily lives. Third, we had the student work for 15% of the session with online interactive tool for two reasons. These tools were initially helpful for the novice programmer to help scaffold them as they expressed their discomfort with development tools. In addition, the student could log-in from home if they missed a session or wanted to revisit the ideas on their own. Fourth, we would work on learning new tools and connecting the concepts through sample programs which were structured at different difficulty levels. The students could work through understanding the programs at their own pace, in accordance to research [14].

3. Evaluating Understandings

We evaluated understanding based on classroom discussions and independently developed and written programs. The students all independently developed a guessing game written in Python on any topic of their choosing. The game included control flow, input, output, data structures, methods and classes. At the beginning of every session, students were asked to recall what they had learned the prior session and would regularly raise their hands to add comments at to what was learned in prior session exemplifying the Pirie and Kieren model [14] of multiple learning layers.

4. Lessons Learned from An After-School Environment

We, as educators and researchers, learned a lot from our fifteen week after school experience. First, we learned that many students had limited, if any, exposure to professional language programming. Second, we found that we did not have adequate system administrator privileges to install software on the student computers. Because of time constraints, machine variations and user system privileges in an after school learning situation, we encourage future practitioners and developers to set-up a virtualized machine environment to streamline consistent code development for non-administrator privileged users. Third, we suggested developing a take-home packet for students with software examples and the same virtual environment on a USB. This may help situations when students do not have anyone at home who can help them learn to program. Fourth, some of our students recommended online code games as alternatives lessons. The students, however, do prefer learning interlaced with sounds, graphics and visualization, as described by Goldwasser [7]; however, many of the game themes are not appropriate for younger students. Fifth, we would encourage a repository of student learning in computing to be constructed as the Video Mosaic Collaborative (VMC) has proven to be quite successful in studying Mathematics Education learnings, as shown by [15]. Finally, our after school curriculum can be downloaded, used and expanded from the *www.technologyinthepark.com/FOE2017/*.

References

[1] Ball, D. L., Hoyles, C., Jahnke, H. N., & Movshovitz-Hadar, N. (2002). The teaching of proof. In L. I. Tatsien (Ed.), Proceedings of the International Congress of Mathematicians, Vol. III (pp. 907–920). Beijing: Higher Education.

[2] C. Kamii and A. Dominck. The Harmful Effects of Algorithms in Grades 1 through 4. The teaching and learning of algorithms in school mathematics: 1998 yearbook. The National Council of Teachers of Mathematics. Reston, VA.

[3] Josh Tenenberg. Industry fellows: Bringing professional practice into the classroom. In Proceedings of the 41st ACM Technical Symposium on Computer Science Education, SIGCSE '10, pages 72-76, New York, NY, USA, 2010. ACM.

[4] Marty Simon. Reconstructing mathematics pedagogy from a constructivist perspective. Journal for Research in Mathematics Education, 26:114-145, 1995.

[5] C. Fosnot and M. Dolk. Young Mathematicians at Work. Heinemann Publishers, Portsmouth, New Hampshire, USA, 2001.

[6] Yankelewitz D. Mueller, M. and C Maher. Promoting student reasoning through careful task design: A comparison of three studies. International Journal for Studies in Mathematics Education, 3, 2010.

[7] Michael H. Goldwasser and David Letscher. A python graphics package for the first day and beyond. SIGCSE Bull., 40(3):326-326, June 2008.

[8] Juha Helminen, Petri Ihantola, Ville Karavirta, and Lauri Malmi. How do students solve parsons programming problems?: An analysis of interaction traces. In Proceedings of the Ninth Annual International Conference on International Computing Education Research, ICER '12, pages 119-126, New York, NY, USA, 2012. ACM.

[9] Jeannette M. Wing. Computational thinking. Communications. ACM, 49(3):33-35, March 2006.

[10] Suzanna Schmeelk. An investigation of fourth grade students growing understanding of rational numbers (Unpublished doctorial dissertation). Rutgers, The State University of New Jersey, New Brunswick, NJ. 2010.

[11] Daniel D. Suthers and Christopher D. Hundhausen. The effects of representation on students' elaborations in collaborative inquiry. In Proceedings of the Conference on Computer Support for

Collaborative Learning: Foundations for a CSCL Community, CSCL'02, pages 472-480. International Society of the Learning Sciences, 2002.

[12] Robert B. Davis. Understanding understanding. Journal of Mathematical Behavior,3 225-41,1992.

[13] Robert B. Davis and Carolyn A. Maher. How Student's Think: The Role of Representations., Mathematical Reasoning: Analogies, Metaphors, and Images.3. Lawrence E. Earlbaum Associates. Hillsdale, NJ. 2003.

[14] Susan Pirie and Thomas Kieren. Growth in mathematical understanding: How can we characterize it and how can we represent it? Educational Studies in Mathematics, 26:165-190, 1994.

[15] Suzanna Schmeelk and Robert Sigley. New Tools for Research: Using the Video Mosaic Collaborative. Proceedings of the American Society for Engineering Educaiton (ASEE). 2012. San Antonio, Texas. 25.975.1--25.275.5.