World Academy of Science, Engineering and Technology
International Journal of Educational and Pedagogical Sciences
Vol:12, No:10, 2018

# Teaching Computer Programming to Diverse Students: A Comparative, Mixed-Methods, Classroom Research Study

Almudena Konrad, Tomás Galguera



***Abstract*****—**Lack of motivation and interest is a serious obstacle to students' learning computing skills. A need exists for a knowledge base on effective pedagogy and curricula to teach computer programming. This paper presents results from research evaluating a six-year project designed to teach complex concepts in computer programming collaboratively, while supporting students to continue developing their computer thinking and related coding skills individually. Utilizing a quasi-experimental, mixed methods design, the pedagogical approaches and methods were assessed in two contrasting groups of students with different socioeconomic status, gender, and age composition. Analyses of quantitative data from Likert-scale surveys and an evaluation rubric, combined with qualitative data from reflective writing exercises and semi-structured interviews yielded convincing evidence of the project's success at both teaching and inspiring students.

***Keywords*****—**Computational thinking, computing education, computer programming curriculum, logic, teaching methods.

## I. Introduction

THE need to develop computational thinking (CT) among students is not a new idea. First introduced by Wing [1], and now viewed as the core of STEM disciplines [2], CT has become an essential skill. Cuny et al. [3] define CT as "the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent" [3]. Leu et al. [4] go beyond a set of skills and view CT more broadly, as "new literacies," or "skills, strategies, and dispositions necessary to successfully use and adapt to the rapidly changing information and communication technologies and contexts".

A historical analysis of the arguments behind literacy programs yields similar perspectives to those arguing in favor of teaching coding: they are beneficial for education, intellectual development, national defense, civic participation, economic productivity, and individual success [5]. However, authors have identified challenges associated with teaching necessary and beneficial coding skills and CT. Specifically, age [6], gender [7], and structured social settings [8] are important variables influencing the development of CT and, by implication, programming among students. This paper considers the computational skills associated with the nature and development of CT as well as programming skills and strategies among children and youth. In addition, computational skills associated with computer programming and its strategies, and how these relate to background variables such as age, gender, and contextual variables are used as a framework for empirical research. This paper also presents conclusions reached regarding curriculum and pedagogy for the development of CT and, specifically, coding skills.

Research has shown that the greatest cognitive gain using educational technology is through simulations and games, rather than traditional programming instruction. This is especially true for female students, and when students have a choice running the games, rather than when teachers controlled the games [9].

Building upon Cuny et al.'s [3] influential definition of CT, the National Science Foundation and the College Board have identified seven "big ideas" in computer science [8]:

1. Computing is a creative human activity;
2. Abstraction reduces information and detail to focus on concepts relevant to understanding and solving problems;
3. Data and information facilitate the creation of knowledge;
4. Algorithms are tools for developing and expressing solutions to computational problems;
5. Programming is a creative process that produces computational artifacts;
6. Digital devices, systems, and the networks that interconnect them enable and foster computational approaches to solving problems; and
7. Computing enables innovation in other fields, including science, social science, humanities, arts, medicine, engineering, and business.

These ideas stress the centrality of abstraction in CT curricula, helping students generalize from specific experiences and deal with complexity [8]. Curricula and materials that introduce students to CT, often allow them first to use and familiarize themselves with the digital environment, inviting them to change it, and finally create new artifacts and related uses [10]. In general, the use of video gaming in curricula and pedagogy for CT development among children has been identified as a potentially powerful instructional approach [11]. Similarly, modeling and simulation that require students to abstract patterns out of observations, develop rules, and apply rules to solve problems represent effective ways to

Almudena Konrad Associate Professor of Computer Science, Mills College, Oakland, CA, 94613, USA (phone: 510 430 2210, e-mail akonrad@mills.edu).

Tomás Galguera Professor of Education Abbie Valley Professorship in Education, Mills College, Oakland, CA, 94613, USA (phone: 510 430 3174, e-mail tomasgs@mills.edu).

engage students [12]. Despite these advances in understanding on how best to teach programming to students, it remains true that students utilize most digital devices as consumers, not creators of content or code [26].

In their review of research on the teaching and development of CT, Kafai and Burke identify three shifts in how youth learn CT: from coding to applications, from tools to communities, and from creating from scratch to remix. They also point to the need to provide youth with "access to participation and collaboration in communities of programming" [26]. Regardless of how one views coding, the expansion of technology in everyday life and preoccupation with preparing a skilled working force [13] demands attention to the development of CT. This paper presents results from one such effort, addressing pertinent implications for research and practice.

## II. Methodology

This study followed a mixed-methods, quasi-experimental design that allowed for a measurement of dimensions of the phenomena studied while including the voices of the participants. The latter also function as useful illustrations for the particular pedagogies and materials used. Furthermore, in utilizing a pre- post-test, quasi-experimental design [14], this study aims to identify specific ways in specific pedagogical or curricular approaches influenced students' understanding of and skills associated with programming for game creation. The choice of mixed-methods design, however, does not represent a particular methodological or philosophical position [15].

### *A. Context of the Study*

The curricula and teaching methods assessed in this paper were developed between 2011 and 2016, for students enrolled in KidsLogic [16], a summer and winter computer programming camp for children and teens. The main goals of these experiential camps were to teach fundamentals of computer programming, logic, and problem-solving techniques, to inspire young students to engage in computer programming, and to identify factors that contribute to the success of this learning process. The program evolved by examining and reflecting on students' learning and modifying methods and curricula accordingly. A comprehensive formal assessment was done during the summer of 2016 on a small group of students.

During the six years, KidsLogic developed a total of five different curricula: Lego Robotics Programming [17], Alice Computer Programming [18], Processing Computer Programming [19], Arduino Robotics [20] and Python [21]. Lego Robotics and Alice camps were for children ages eight to 12 years, with beginner, intermediate and advanced lessons. Camps relying on Processing, Arduino, and Python were for children up to 15 years old. Both camps included materials designed to be inclusive of children with different backgrounds and learning abilities in groups of between 10 and 16 students. Each section had at least one instructor and a teacher assistant with a background in computer programming.

Fig. 1 shows the total enrollment for each of the winter and summer programs offered in the six-year period. Most students who enrolled in KidsLogic summer camps were boys; recruiting girls has been a challenge, despite purposefully recruiting girls and providing incentives, including reduced or free tuition. Over the six years, 28 percent of all students have returned to the camp at least once. Of these, four girls have attended six or more sessions; two boys have attended six sessions.

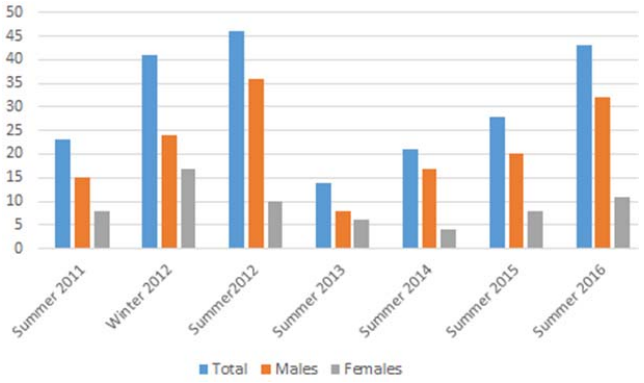


Fig. 1 KidsLogic Yearly Enrollment

In the summer of 2016, the opportunity arose to teach a programming course to first-generation students entering a women's liberal arts college. Although this was the first time that the curriculum would be taught to older students from different backgrounds from those of KidsLogic camps, it provided a setting for assessing the curricula and pedagogy with a different student group. Further, given the challenges in recruiting female students for the KidsLogic camp, the prospect of teaching programming to an all-female group of students mostly of ethnic and linguistic minority backgrounds offered the opportunity to test the educative power of the Python-based tasks and compare outcomes with those of the summer camp.

Research has shown that simply providing access to computers will not bridge the digital divide between privileged and non-privileged students. Rather, it is working collaboratively and creatively, especially engaged in programming, what makes a difference for students [25]. Teaching collaborative programming tasks and materials to first-generation, female, mostly minority students offered an excellent opportunity to replicate this research.

### *B. Study Design*

This exploratory study relies on mixed methods for data collection and analyses. On the first day of the camp, students were asked to complete a 23-item survey that included 14 open-ended, short-answer items and eight, 5-point Likert scale items. Four of the open-ended items requested background information, such as reasons for attending the camp, frequency of and purposes for using digital devices, and favorite academic subject. The remaining ten open-ended items as well as the nine Likert scale items (19 total items) assessed

students' familiarity with CT concepts, skills, and procedures. At the end of the camp, students completed the 19-item assessment component of the survey.

In order to evaluate the students' programming skills, the students worked on a final project in teams of two, where each team had to design and implement a Python computer game. While designing and building their final game, students read and followed a rubric as a guideline, intended to help them focus on skills they needed to demonstrate while creating a game that was original, interactive, designed in graph paper, and error free. In addition, the code needed to successfully implement (a) Boolean variables, (b) 'while' statements, (c) 'for' loops, (d) 'if-else' statements, (e) random functions, (f) new functions, and (g) math operation on variables.

Two focal students who had demonstrated contrasting skills during the camp were selected from each group for semi-structured interviews conducted by the researchers. The purpose of the interviews was to understand the students' experiences completing tasks, following the curriculum and methods, and working with related content. Transcriptions of digital recordings of the interviews were analyzed with other data.

### *C. Participants*

This study relies on findings drawn from the data gathered from two separate groups of participants. The first group consisted of students ranging in ages between eight and 15 years enrolled in a one-week-long (20 hours total) KidsLogic, Python Computer Programming summer camp. KidsLogic camp participants came from relatively high socioeconomic status families, diverse academic skills, and with minimal or very limited computer programming skills. Data were gathered from ten KidsLogic students, six boys and four girls; five boys were returning students.

The second group consisted of students enrolled in a summer workshop (SW) at a women's liberal arts college. The four-week program is specifically designed for incoming, first-generation students, most of whom are women of color. This program strives to ease the transition to college, offering classes in subjects such as English, Sociology, Social Justice and Technology. The courses emphasize writing, communication, and mathematical skills as well as providing students with opportunities to establish support groups, while familiarizing themselves with the physical and social aspects of life on a college campus. In the summer of 2016, a total of 20 SW students participated in a three-day (12 hours total) Python computer programming workshop. This group of female students experienced the same curriculum and pedagogy than the KidsLogic group. However, the total classroom time was 10.5 hours for the SW group and 17.5 hours for the KidsLogic group. Python computer programming was new to all participants in both groups.

### *D. Methods*

Instruction days for both KidsLogic and SAW participants were divided in two parts: (1*) Read*, *Type, Execute and Learn*, and (2) *Design, Implement, Test and Debug*. The first part of the day emphasized learning new concepts through reading and typing code. Reading or "tracing code" [23] is an essential programming skill, requiring at least 50% accuracy in order to write code with confidence [24]. The second part of the day allowed students to try out what they had previously learned, while testing and debugging.

The instructor began by demonstrating to the entire class a computer game written in Python, projected on a screen. The chosen computer game for each session contained purposefully chosen computer skills and concepts. Next, working in pairs and practicing pair programming, students transferred the python code from a handout to the PyCharm IDE (Integrated Development Environment) [22]. Students ran and played with their games, while practicing reading. Meanwhile, the instructor and an assistant circulated around the classroom, making observations, answering questions and providing help as needed.

Toward the end of the first part of the day, and as a group, the students and instructor reviewed and explored the code, writing down new programming concepts, skills learned, and terminology or definitions. Depending on the difficulty of the program and the proficiency of students, it was possible for particular pairs of students to read and type from one to three programs per day.

The goal for the second part of day was for students to build their own computer game. In pairs, students practiced the four steps involved in creating a new Python game:

1. Design the scene of a game and its elements on graph paper;
2. Write simple algorithms to solve problems whenever needed;
3. Implement the game in the PyCharm IDE using the Python language;
4. Test the game, debugging the program to make it error free.

When problems arose, students referred to their designs on chart paper for reference, identifying problems and possible solutions under the guidance of the instructor and assistant. Students presented their finished programs to the instructor for verification only when they were confident it worked as designed.

During assessment days, the instructor provided a total of seven Python games as examples, each one emphasizing particular concepts, such as variables, print statements, function calls, if-else statements, while statements, RGB (Red Green Blue) colors, simple array, function creation, Boolean variables, for-loops, and event driven programming. During the entire camp, the instructor repeated these concepts and explanations, emphasizing ways in which students had implemented these differently. The main goal was to learn through meaningful practice, while remaining engaged and enjoying the tasks.

At the end of assessment days, and utilizing paired programming, students collaborated to develop final Python computer game projects. Five teams from the KidsLogic camp and 10 from the SW presented their final project to their classmates. They were required to submit the game design on

graph paper, written algorithms used, and the game as a digital file. A 12-criterion rubric was used to assess their work.

## III. Results

What follows is a summary of results emerging from the surveys, the semi-structured interviews with the eight focal students (four each from KidsLogic and SAW), and the evaluation of the students' final projects. Results are presented in order of relevance and magnitude.

### *A. Survey Responses*

As mentioned, the Python Computing Programming Assessment Instrument survey contained a combination of eight Likert-scale items and 14 short-answer items, yielding quantitative and qualitative data, respectively. The combination of item types made it possible to test the internal validity of the instrument, whereas the pre- and post-test design allowed assessment of learning resulting from the summer camp and the workshop for first generation college students.

Student t-tests were performed on the pre- and post-test self-evaluations for key concepts, with statistically significant differences emerging in both the KidsLogic and SW groups. Tables I and II contain the scores for each of the matched pairs of pre- and post-test scores in each group, respectively. As can be seen, the improvements in self-assessed knowledge of the nine vocabulary words were statistically significant ($p \leq 0.05$) for all but Variable. There were also differences as to the specific words for which each group developed a greater understanding. KidsLogic students' growth in understanding was greatest regarding Algorithm, Print Statement, Boolean variable, and Loop Statement. In contrast, SW students' understanding grew most with Computer Function, Random Function, Conditional Statement, and Interactive Program.

TABLE I
KidsLogic Pre- Post-Test Paired Samples Student T-Test

| Pre-Post Pairs | Mean Diff. | Std. Dev. Diff. | Std. Error Mean Diff. | t | df | Sig. (2-tailed) |
|---|---|---|---|---|---|---|
| *Algorithm* | -1.55 | 0.19 | 0.06 | 3.40 | 10 | 0.0068 |
| *Variable* | -0.82 | 0.50 | 0.15 | 2.17 | 10 | 0.0552 |
| *Print Statement* | -2.09 | 0.28 | 0.09 | 4.80 | 10 | 0.0007 |
| *Boolean Variable* | -2.45 | 0.47 | 0.17 | 7.22 | 10 | 0.0001 |
| *Loop Statement* | -3.09 | 0.42 | 0.15 | 8.40 | 10 | 0.0001 |
| *Computer Function* | -1.09 | 0.12 | 0.04 | 4.35 | 10 | 0.0014 |
| *Random Function* | -1.82 | 0.19 | 0.06 | 4.10 | 10 | 0.0021 |
| *Cond Statement* | -2.45 | 0.03 | 0.01 | 5.18 | 10 | 0.0004 |
| *Interactive Program* | -1.64 | 0.30 | 0.09 | 3.62 | 10 | 0.0047 |

TABLE II
SW Pre- Post-Test Paired Samples Student T-Test

| Pre-Post Pairs | Mean Diff. | Std. Dev. Diff. | Std. Error Mean Diff. | t | df | Sig. (2-tailed) |
|---|---|---|---|---|---|---|
| *Algorithm* | -0.74 | 0.15 | 0.04 | 3.24 | 18 | 0.0045 |
| *Variable* | -0.58 | 0.18 | 0.04 | 1.93 | 18 | 0.0689 |
| *Print Statement* | -1.68 | 0.69 | 0.16 | 4.16 | 18 | 0.0006 |
| *Boolean Variable* | -1.42 | 0.99 | 0.23 | 4.75 | 18 | 0.0002 |
| *Loop Statement* | -2.00 | 0.23 | 0.05 | 5.85 | 18 | 0.0001 |
| *Computer Function* | -1.63 | 0.09 | 0.02 | 6.37 | 18 | 0.0001 |
| *Random Function* | -2.79 | 0.34 | 0.07 | 12.45 | 18 | 0.0001 |
| *Cond Statement* | -2.05 | 0.56 | 0.13 | 5.93 | 18 | 0.0001 |
| *Interactive Program* | -1.63 | 0.41 | 0.09 | 6.68 | 18 | 0.0001 |

Students' responses to the nine open-ended items that were included in the Python Computer Assessment Survey allowed for triangulation and a test of internal validity of the quantitative pre-post results. A crucial survey item, '*What steps would you follow to create a digital game?'* was especially useful to determine the extent to which students' CT grew during the duration of the KidsLogic summer camp and the SW workshop. From the KidsLogic group, all 10 students were unable to produce correct responses for the digital game question in the initial assessment. Yet, in the post-test responses, all students produced correct answers. Further, the short-answer responses students wrote commonly mention needing to find a solution, creating an algorithm, implementing the solution on Python, and testing and debugging it. These are all accepted and recommended programming steps. In contrast, only six of the 19 SAW students (31%) mentioned these steps in their last day evaluation survey responses.

Logic, an essential concept in computer programming, was another concept that the students were asked to describe in a short answer survey item. Initially, only a few KidsLogic students and no SW students wrote a correct definition. The responses for both groups improved by the end of the camp. Many students defined logic as the way computers think and were able to write a logical if-else statement.

### *B. Interview Responses*

All eight students selected for the interviews were not only willing to describe their experiences, but also spoke candidly about their observations as KidsLogic or SW students. Their

statements indicated a general enjoyment of the tasks they were asked to complete as well as the steps they followed. Students in both groups mentioned specific ways in which the camp had been beneficial, although SW students also mentioned initial apprehension and resistance to the idea of attending a coding workshop. The following are examples of such sentiments:

> I never had a formal class in programming. I am learning a lot [...] This is a cool opportunity. I think coding is cool [Female KidsLogic student]
>
> Everything I have learned about programming has been through this camp. I got to freely work on my own ideas but we also got help. Now I can make a simple console game. All I learned was new. (Male KidsLogic student)
>
> Before this workshop I didn't know how much work goes into making simple images, or making something moving. The coding is amazing! So much work goes into it. I learned to be patient. I like that we had so many helpers, and even helpers are trying to figure out things. This is new to me. I don't do computer stuff; all has been challenging. Coding seemed so complex! This is all completely new to me. All new language for me, but we are getting the hang of it (Female SW student).
>
> I am not technology oriented; this is my first exposure to programming. I was not interested in computers in high school" (Female SW student)

Both groups found debugging to be the most challenging step in the process; however, they also mentioned experiencing satisfaction when they were able to figure out the problem. As such, students not only learned specific skills and information specific to Python but also problem solving strategies in general. Moreover, working with a classmate seemed to have contributed to the participants' resiliency and social skills as well. The following quote illustrates several of these outcomes:

> When you have a problem with a person, you talk it out and get some type of agreement. With computers you try to fix it, you look at the history to see what is what you did to cause the problem. With a person, you confront the person. With technology, you look at the history. For programming, I do research and ask people. Humans keep learning and adding information. But computers don't. In this camp, I learned the Boolean statements, True or False, like a flag, I never heard about it before (Female KidsLogic student)

Python proved to be an accessible and sufficiently flexible programming language for both age groups. Also, the students reported finding the paired work helpful in that they were able to take turns trying out skills and strategies, but even more importantly, a second pair of eyes helped them avoid mistakes and spot keystroke errors. Students also appreciated the impromptu explanations and mini-lectures used when the instructor addressed a common problem or misconception.

> [What helps is] that you have a driver and an observer, the fact that you are not always typing, and you get ideas from other people. More people working together means more possibilities. Too many people then you don't get anything done. But two people, I like that. It is always a good idea to work with a partner, even if you disagree. You get to know the person based on the ideas they have. (Female KidsLogic student)
>
> Working with a partner is more helpful, we try to solve the problem together. Two brains work better than one. [Coding] is really precise. I have done HTML before, and this was a struggle. On my own, it didn't work, but now with this group I feel better. It is very rewarding; we try to go deeper together. I like that the professor stops the class for a few minutes to give short mini lessons. It is helpful. I knew about RGB colors, but the rest was new. The beginning of troubleshooting was frustrating. It is not as frustrating now. I keep debugging and I get something out of it, and I want to do more (Female SW student)

One of the SW first-generation student was a bilingual Latina immigrant who, though cautious and hesitant at first, found the workshop quite useful in a variety of ways. An unexpected result was that this student enrolled in a fall semester computer sciences course, immediately after the workshop. This was also true of five other students who enrolled in the course as well. Below is the immigrant student's quote (translated from Spanish):

> I am from Mexico and have lived in the USA for five years. This class has changed my opinion about computers. Before, I didn't like them. Now I know that programming can achieve a variety of things like building video games. One can also build things for big companies. I have learned about RGB and am able to put images in video games. I can make the images move. I didn't know anything about programming! I learned from making mistakes and this opened my mind. To find errors is frustrating. I now understand that computer programming is not that bad. I want to take more of these classes. I would have like the workshop to run longer (Female SW student).

### *C. Final Project*

The evaluation of final projects yielded generally satisfactory results, with scores ranging from a low of 12 to a high of 19 (out of 20 possible). KidsLogic teams' mean score was 15.4 (SD=2.7), whereas SW teams' mean score was 15 (SD=2.0). The team with the highest score (19) was a SW team, with a KidsLogic with two 15-year-old girls scoring 18, or second highest. The former had no previous programming skills, while the latter had enrolled in 10 previous KidsLogic camp sessions, neither had previous experience with Python.

## III. Conclusion

The six-year project reported in this paper has succeeded at creating pedagogy and curricula that (1) inspire and motivate youth of different backgrounds and interests to develop CT; (2) teach fundamentals of computer programming to youth, and especially young women; and (3) learn from the process on how to engage students in collaboration and learning. The

program succeeded at motivating students to continue learning programming and developing their CT beyond the workshop. This is particularly remarkable given that students in both groups were required to attend the course either by their parents or college counselors. As such, it is not possible to generalize the results from this study to contexts in which students choose to enroll in the course. Beyond the evidence captured by the multiple data gathering methods described in this paper, the strongest evidence of the effectiveness of the pedagogy and curricula are the spontaneous behaviors exhibited by a large portion of the students in each group after their respective courses. Several KidsLogic former students have joined or started computer and robotics clubs at their respective schools. And more than one-fourth of all SW students enrolled in introductory computer science courses at their university.

The t-tests results are evidence that the content and pedagogy led to growth in understanding important computer programming concepts in both groups of students. Furthermore, students were able to define and understand the steps involved in designing and building computer programs. Even students who were unable to clearly and precisely name the steps, they were still able to follow the steps during practice hours. Similarly, at the end of the camp, most students were not able to fully understand, much less define, computing logic as a central element of programming and CT. Nevertheless, all students were able to write a logic statement, having learned to read logic statements by constantly tracing code provided by the instructors.

Generally speaking, students were motivated to learn by providing them with interesting working pieces of code, and by working backwards, from a finished product, toward an understanding the code in a collaborative setting. Students were very motivated to understand what parts of the code were responsible for certain actions in the game. This curiosity was the catalyst for learning, while the enjoyment of the tasks provided motivation. KidsLogic curriculum was created with an effort to present programs and games to students in accessible and engaging ways. At the end, both groups of students created fun games without even thinking about the complexity and logic in their code.

## REFERENCES

[1] Wing, J. (2006). Computational thinking. Communications of the ACM, 49(3), 33–36.

[2] Henderson, P. B., Cortina, T. J., Hazzan, O., and Wing, J. M. (2007). Computational thinking. In Proceedings of the 38th ACM SIGCSE Technical Symposium on Computer Science Education, (SIGCSE '07), 195–196. New York, NY: ACM Press.

[3] Cuny, J., Snyder, L., & Wing, J.M. (2010). Demystifying computational thinking for non-computer scientists. Unpublished manuscript in progress, referenced in http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf.

[4] Leu, D. J., Kinzer, C. K., Coiro, J. L., & Cammack, D. W. (2004). Toward a theory of new literacies emerging from the Internet and other information and communication technologies. In R. B. Ruddell, & N. J. Unrau (Eds.) Theoretical models and processes of reading (5th ed.) (pp. 1570-1613). Newark, DE: International Reading Association.

[5] Vee, A. (2017). Coding Literacy: How Computer Programming is Changing Writing. MIT Press.

[6] Grandell, L., Peltomäki, M.,Back, R. J., & Salakoski, T. (2006, January). Why complicate things?: introducing programming in high school using Python. In Proceedings of the 8th Australasian Conference on Computing Education-Volume 52 (pp. 71-80). Australian Computer Society, Inc.

[7] Mercier, E. M., Barron, B., & O'Connor, K. M. (2006). Images of self and others as computer users: The role of gender and experience. Journal of Computer Assisted Learning, 22, 335–348. San Francisco Unified School District (n.d.). In Computer Science for All Students in SF. Retrieved July 1st, from http://www.csinsf.org/curriculum.html.

[8] Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. Educational Researcher, 42(1), 38-43.

[9] Vogel, J. J., Vogel, D. S., Cannon-Bowers, J., Bowers, C. A., Muse, K., & Wright, M. (2006). Computer gaming and interactive simulations for learning: A meta-analysis. Journal of Educational Computing Research, 34(3), 229-243.

[10] Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Werner, L. (2011). Computational thinking for youth in practice. ACM Inroads, 2, 32–37.

[11] Holbert, N. R., & Wilensky, U. (2011, April). Racing games for exploring kinematics: a computational thinking approach.

[12] Blikstein, P. (2010). Connecting the science classroom and tangible inter- faces: the bifocal modeling framework. In Proceedings of the 9th International Conference of the Learning Sciences, Chicago, IL, 128–130.

[13] Berrett, D. (2015). The day the purpose of college changed. The Chronicle of Higher Education, 26.

[14] Harris, A. D., McGregror, J. C., Perencevich, E. N., Furuno, J. P., Zhu, J., Peterson, D. E. & Finkelstein, J. (2006) The use and interpretation of quasi-experimental studies in medical informatics, 13(1), 16–23.

[15] Maxwell, J. A. (2016). Expanding the history and range of mixed methods research. Journal of Mixed Methods Research, 10(1), 12-27.

[16] KidsLogic http://www.kidslogic.net/.

[17] LEGO Mindstorms https://www.lego.com/en-us/mindstorms/?domainredir=mindstorms.lego.com.

[18] Alice http://www.alice.org/.

[19] Processing https://processing.org/.

[20] Arduino https://www.arduino.cc/.

[21] Python https://www.python.org/.

[22] PyCharm https://www.jetbrains.com/pycharm/.

[23] Sentance, S. & Csizmadia, A. Educ Inf Technol (2017) 22: 469. https://doi.org/10.1007/s10639-016-9482-0.

[24] Lister, R. (2011). Concrete and other neo-piagetian forms of reasoning in the novice programmer. Proceedings of the Thirteenth Australasian Computing Education Conference - Volume 114, Perth, Australia. 9–18.

[25] Warschauer, Mark; Matuchniak, Tina. New Technology and Digital Worlds: Analyzing Evidence of Equity in Access, Use, and Outcomes, Review of Research in Education, v34 n1 p179-225 2010.

[26] Kafai, Y. B., & Burke, Q. (2013). Computer programming goes back to school. Phi Delta Kappan, 95(1), 61-65.