Assessing Computational Learning in K-12

Shuchi Grover
Graduate School of Education
Stanford University
Stanford, CA 94305
shuchig@stanford.edu

Stephen Cooper
Computer Science Department
Stanford University
Stanford, CA 94305
coopers@stanford.edu

Roy Pea
Graduate School of Education/
H-STAR Institute
Stanford University
Stanford, CA 94305
roypea@stanford.edu

ABSTRACT
As computing curricula continue to make their way into K-12 schools, the issue of assessing student learning of computational concepts remains a thorny one. This paper describes the multiple forms of assessments used in a 6-week middle school curriculum with the goal of capturing a holistic view of student learning. A key aspect of this research is the use of instruments developed and shared in prior research. Included among these were several questions used in an Israeli nationwide exam to test middle school student learning of programming in Scratch. This paper reports on the use of the curriculum in two studies conducted in a public US middle school classroom, and compares performances of these students with those reported by the Israeli Ministry of Education in their large-scale study. It also argues for multiple modes of assessment of computational learning in K-12 settings.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education - *Computer Science Education, Curriculum*

General Terms
Design, Experimentation, Human Factors.

Keywords
Computational Thinking, Computer Science Education, Computing education, assessment, K-12 curriculum development.

1. INTRODUCTION
Computational Thinking (CT) is recognized as a necessary skill for today's generation of learners [26]. A consensus has been building around the view that all K-12 children must learn CT [10] and be offered experiences with computer science. Several recent efforts are underway among researchers and educators working in concert with organizations such as CSTA and NSF to define guidelines for–and designs of K-12 (especially high school)–curricula. Many of these introductory experiences are being designed in the context of programming in block-based environments such as Scratch, Alice, and MIT App Inventor. Despite the many efforts aimed at tackling the issue of CT assessment [12,17,25], there are several challenges for assessing the learning of computational concepts and constructs in these programming environments.

Without attention to rigorous assessment, CT can have little hope of making its way successfully into K–12 school education settings at scale [9]. Our work on assessment of computational thinking is inspired by–and builds on–noteworthy efforts described in the next section that have attended to assessment of computational learning (CL) [6] in the context of Scratch and Alice. It also draws on recent work on deeper learning and the need to build and assess core disciplinary knowledge and students' ability to transfer conceptual learning, in addition to interpersonal and intrapersonal abilities [20].

2. RELATED WORK
2.1 Assessment of Computational Learning
In the context of block-based programming, there have been few studies devoted specifically to assessing foundational CT concepts like algorithmic thinking, repetition and selection in the algorithmic flow of control (loops and condition?al logic). Among these are a series of investigations in the context of game programming in Alice with middle school students within school and afterschool settings [25]. These studies were conducted with the aim of providing motivating experiences in computing contexts to empower students from underrepresented communities. The Alice "Fairy Assessment" requires students to code parts of a predesigned program to accomplish specific tasks. By having students modify or add methods to existing code, the researchers assessed student understanding of algorithms, abstraction and code. This assessment is Alice-based and requires subjective and time-consuming grading, a challenge for assessing student code.

Brennan & Resnick [5] highlight issues related to assessment of CT, especially with grading student-created programs. They underscore the need for multiple means of assessment. Student-created artifacts while "rich, concrete and contextualized" and a necessary tool for assessing students, do not tell the whole story of student understanding. They lack elements of process and are often misleading indicators of student understanding [21]. Though student projects point to apparent fluency as evidenced by the existence of certain computational constructs in the code, probing deeper through questions may reveal a different story. When asked about how parts of their code work, students' descriptions often reveal significant conceptual gaps, as they cannot explain how their code works. It is salient to note a similar observation in [25] where the authors discuss "students' partial understanding of someone else's code" in their analysis of the results of student performance on the Fairy Assessment mentioned above. This problem may occur when students work in pairs or when the learning environment allows for students to seek and give help, or copy and paste code. Assessing student projects is also subjective and time-consuming, especially with a large student population.

There is a need for more objective assessment instruments to illuminate student understanding of specific computing concepts and other CT skills such as debugging, code-tracing, problem decomposition and pattern generalization. Cooper [18] created a multiple-choice instrument for measuring learning of Alice programming concepts, but it has not been used to measure student learning in K-12 education. Lewis' online Scratch course for middle school [14] also uses such "quizzes".

It is also important that students learn—and be assessed on—the vocabulary of computing. Analogous to the benefits seen in developing and using academic vocabulary in science and math [13], fostering deeper computational learning and an affinity for CS includes building a language of the domain to aid thinking about and communicating computational ideas more effectively and learn the shared vocabulary of the discipline and its community of people.

This belief is seen in studies conducted in Israel involving systematic assessments of students' conceptual vocabulary and CL in the course of a semester-long class for introducing computer science concepts to 9th graders [17]. Their work also tackles aforementioned issues related to relying on student-created artifacts for assessing CL. They used pre-, interim and post-tests designed to assess CT through a combination of Bloom's modified taxonomy as well as the SOLO taxonomy [3]. They tested students on CT terms, and also required them to solve problems related to a pre-designed Scratch programming task. These were done on paper in response to scripts presented in the test and through additions to existing code in Scratch in response to question prompts. The tests were thus more objective and able to assess a student's use of appropriate computational constructs in a pinpointed way. Ben-Ari generously shared the instruments with the lead author via an email exchange.

2.2 Large-scale Efforts

A few large-scale efforts to roll out introductory computing curricula at the middle school level include useful ideas for assessment. Prominent among these are the UK national effort [23] and the Israel Ministry of Education's Science and Technology Excellence Program including a national curriculum and exam [27]. The UK curriculum includes objective exercises using Scratch code, Scratch programming assignments and a final project of the student's choosing. We have used all three modes of assessment in our curriculum as well.

It is the use of multiple-choice assessments and attendant rubrics to measure learning in the Israeli effort [28] that make their work particularly pertinent. Their national exam comprises 9 questions (with roughly 30 sub-questions). Arguably such multiple-choice measures are easier to implement in a large-scale setting than open-ended student projects. Similar to previous work done in Israel [17], they use Bloom's taxonomy to classify the questions and inferences that can be drawn about the appropriate learning level of the associated computing concepts. The assessment tool was used to measure not only student learning but to evaluate the curriculum that the Ministry hopes to evolve based on results. This highlights a critical, but often-ignored purpose of assessment— *leveraging assessments to improve a curriculum so as to better meet learning goals*.

With the stated focus on deeper learning of computational concepts, our research draws on all these ideas of multiple assessment mechanisms or a "system of assessments" [7] to assess to deeper learning in a computing context. We thus designed a curriculum with structured formative and summative assessment instruments in addition to programming assignments and open-ended projects in Scratch. By reusing assessment instruments from prior efforts, we also test their use in new settings. For example, we employ many of the same questions used in the Israel national exam, and their grading rubrics. Beyond using these assessments to measure student learning in a holistic way, our design-based research effort also used them to refine our curriculum over two iterations. The following sections describe our iterative efforts involving our introductory CS course for middle school using Scratch that not only employed several types of formative and summative assessments based in Scratch, but also included novel assessments that measured learners' ability to transfer those skills to a text-based programming context. The latter are described in [11]. We describe the assessments used, and in the results and discussion section, provide a comparative report between the results from Israel and ours. We comment on what we believe makes our effort distinct from prior studies.

3. METHODOLOGY

This section describes design-based research involving two studies of a six-week middle school module, titled ***"Foundations for Advancing Computational Thinking" (FACT)***. The module was designed to include elements aimed at building awareness of computing as a discipline while promoting engagement with foundational computational concepts such as algorithmic flow of control comprising sequence, looping constructs, and conditional logic. The goal of the research was to study multiple and novel mechanisms for assessing learning of these core computational concepts, and helping to refine the curriculum.

3.1 Curriculum Design

As a short, introductory module, we believed students would be well served with material focused on the most basic CT topics. Our module focused on systematic processing of information, structured problem decomposition, algorithmic notions of flow of control including selection and repetition (i.e. conditional logic and iterative thinking) along with some learner engagement with abstractions and pattern generalizations as well as debugging. The organization of the 6-week module is shown in Table 1. Each topic mapped roughly to a week of course contact time.

The curriculum adopted the following approaches in its design:

- Builds on the rich body of prior research involving children and novice programmers to guide the pedagogy and assessments for the content being taught. These include: using worked examples for conceptual learning [24], using pseudo-code [4], teaching reading/code-tracing [15], and using frequent multiple-choice "quizzes" to push student understanding and reinforce concepts learned [9];
- Makes explicit the foundational ideas of computer science and computational thinking [16];
- Uses academic language to explain concepts in terms of the vocabulary of the computer science domain [13]; and
- Promotes active, constructivist learning in Scratch through several hands-on activities and assignments.

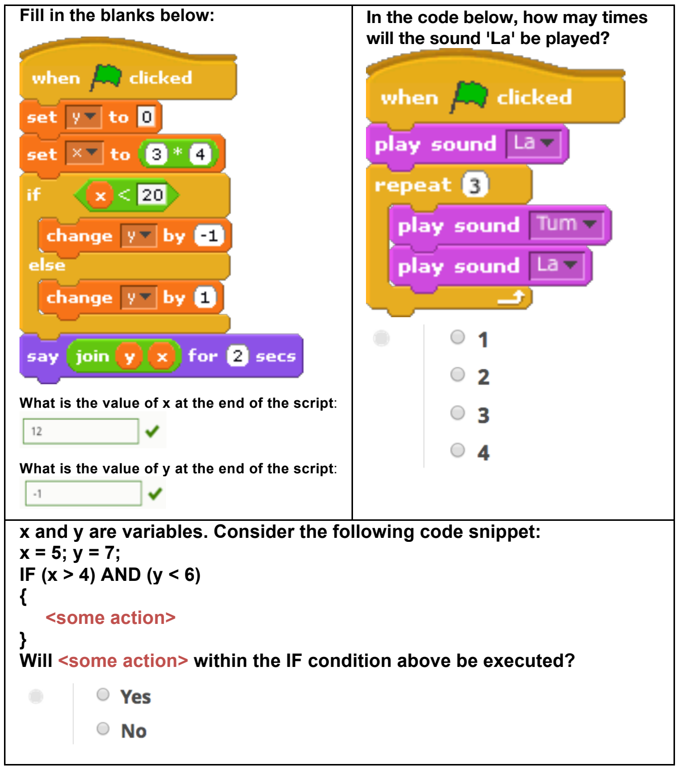
Table 1: Topics covered in 6-week FACT Intro CS module

Unit 1	Computing is Everywhere! / Algorithms / Programs
Unit 2	Serial execution; Problem solving, task breakdown, solution as precise sequence of instructions
Unit 3	Iterative/repetitive flow of control: Loops
Unit 4	Data and variables
Unit 5	Boolean Logic & Advanced Loops
Unit 6	Selective flow of control: Conditional thinking

3.1.1 *Design of Formative Assessment of CL*

Formative assessment was integrated throughout the course as multiple-choice quizzes, designed to give learners encouraging feedback and explanations. Many quizzes included small snippets of Scratch code on which questions were based. These were similar to those used in existing curricula [14,23]. These assessments aimed to help learners develop familiarity with code tracing and the ability to understand an algorithm in Scratch or in pseudo-code [4,15]. Figure 1 shows three sample quiz questions. Correct answers to quiz questions were accompanied by explanations. Some formative assessments also involved presenting jumbled blocks in Scratch required for a program (akin to Parson's puzzles), and having students snap them in correct order [19].

The curriculum placed a heavy emphasis on learning by doing in Scratch. In addition to open-ended time to dabble in Scratch, there were specific assignments that built on the concepts taught in the preceding lecture(s). Sample assignments include making a "spirograph" (using nested loops); a polygon generator depending on a size and shape specified by the user (employing user inputs and variables); "4-quadrant art" which colors the screen in different colors depending on the position of the cat (using conditionals with compound Boolean conditions); 2-paddle pong (using conditionals within repeat-until loops); "guess my number" game (which uses all the constructs taught through the course). *The assignments were manually graded based on rubrics provided to students*. Students had the freedom to design specific elements of their artifacts or add to them.



Fill in the blanks below:

```
when 🚩 clicked
set y to 0
set x to 3 * 4
if x < 20
    change y by -1
else
    change y by 1
say join y x for 2 secs
```

What is the value of x at the end of the script: 12 ✓

What is the value of y at the end of the script: -1 ✓

In the code below, how may times will the sound 'La' be played?

```
when 🚩 clicked
play sound La
repeat 3
    play sound Tum
    play sound La
```

- 1
- 2
- 3
- 4

x and y are variables. Consider the following code snippet:

```
x = 5; y = 7;
IF (x > 4) AND (y < 6)
{
    <some action>
}
```

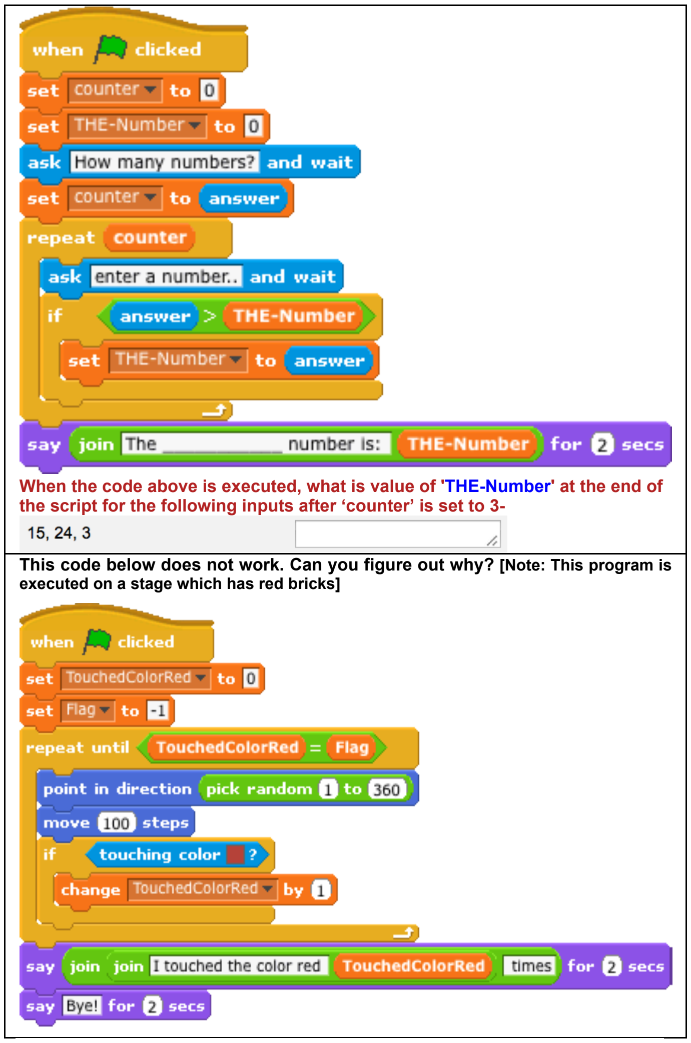
Will <some action> within the IF condition above be executed?

- Yes
- No

Figure 1: Sample quiz questions used in formative assessments

3.1.2 *Design of Summative Assessments of CL*

The pre- and post-test instruments borrowed from earlier work in Israel [17,28], both of which used the Scratch environment. The Israeli national curriculum [28], like ours, focuses on task decomposition, sequences, loops and conditionals as the foundational building blocks of algorithmic thinking. This made it convenient to reuse questions from their exam. We incorporated questions 2, 4, 5, 7, 8 and 9 from their national exam [28]. We did not incorporate the other three questions since we did not get access to details on those questions prior to our study's launch. In addition to the six questions from the national exam, we required students to provide definitions of key computational terms such as *algorithm*, *variable*, *initialization*, *conditional*, *Boolean variable*, and *loop*. These were borrowed from [17] as were some additional questions to test student understanding of algorithms written in pseudo-code. We also included questions that used snippets of basic Scratch code to test if students could identify the core constructs in them as used in [8]. Lastly, we added a few questions of our own to assess code-tracing and debugging skills in snippets of code that used more advanced looping and conditional logic, as shown in Figure 2.



```
when 🚩 clicked
set counter to 0
set THE-Number to 0
ask How many numbers? and wait
set counter to answer
repeat counter
    ask enter a number.. and wait
    if answer > THE-Number
        set THE-Number to answer
say join The ________ number is: THE-Number for 2 secs
```

When the code above is executed, what is value of 'THE-Number' at the end of the script for the following inputs after 'counter' is set to 3-

15, 24, 3

This code below does not work. Can you figure out why? [Note: This program is executed on a stage which has red bricks]

```
when 🚩 clicked
set TouchedColorRed to 0
set Flag to -1
repeat until TouchedColorRed = Flag
    point in direction pick random 1 to 360
    move 100 steps
    if touching color 🟥 ?
        change TouchedColorRed by 1
say join join I touched the color red TouchedColorRed times for 2 secs
say Bye! for 2 secs
```

Figure 2: Questions added to our post-test in our studies (in addition to questions from Israel)

3.2 Study and Data Measures

3.2.1 *Participants & Procedures*

For Study 1, the FACT module was taught for six weeks in April-May, 2013 in a public middle school classroom in Northern California. The student sample comprised 26 children from 7th and 8th grade (21 boys and 5 girls, mean age: ~13 years) enrolled in a semester-long "Computers" elective class. The course was taught face-to-face in a computer lab (lectures and demonstrations in Scratch), with the units on conditionals and Boolean logic offered online in the form of short videos. The online units were done as a pilot to get student feedback ahead of online deployment of the entire module.

For Study 2, the FACT curriculum was taught in Sept-Oct, 2013 in the same middle school as Study 1, but with a new cohort of students in the "Computers" elective class (20 boys, 8 girls, mean age: 12.3 years). Study 2 involved the use of a completely online version of the course deployed on the Stanford OpenEdX online platform. The lectures and Scratch demonstrations in this version of FACT were in the form of short Khan Academy-style videos ranging between 1-6 minutes in length. Some changes were made to the assignments and duration for which certain concepts were taught based on our experiences, student performance, and student feedback from Study 1. For example, we devoted more time to loops and variables; more projects involving games and art were incorporated; and a more formal final project requirement was added with student interviews based on the project. The quizzes described in section 3.1.1 used automated grading and feedback in OpenEdX in Study 2. In Study 1 these were administered through Schoology, a learning management system used by the school, which also allowed for auto-grading and feedback. In both studies, the class met for 55 minutes four times per week. The lead researcher on this effort was also the curriculum developer and teacher for the FACT module. An independent researcher assisted with subjective grading.

3.2.2 Data Measures

In both studies, data were captured for assessing the curriculum and student learning:

- *Prior Experience Survey*: These gathered information about students' prior experiences in computational activities, especially programming [1]. This data was used for regression analyses to explain variances in student performance that are beyond of the scope of this paper.
- *Pre-Test***:** This measured prior knowledge of computational concepts, including questions on the definitions of computing terms, student understanding of serial execution of algorithms presented in English, and questions that tested student knowledge of Scratch and programming in general. These were borrowed from [8,17].
- *Post-Test*: This measured student knowledge of computational concepts, the ability to read and decipher code or pseudo-code, and to debug a piece of code. The focus of these tasks was on algorithmic flow of control: sequence, loops, and conditionals. As described in 3.1.2, the post-test included questions from [8,17,28] as well as those we created. The pretest was a subset of the post-test as it seemed unreasonable to give children a large number of problems on skills and content completely alien and new to most students.
- *Quizzes:* Although not used to "test" students, data on student performance in these formative assessments were gathered as indicators for monitoring student progress and capturing conceptual targets of difficulty.
- *Scratch Assignments:* These were given throughout the course and graded according to a rubric.
- *Final Scratch Projects & Student Interviews (only in Study 2):* Students used the final project-guiding document used in the UK curriculum [23] for planning and reflecting on their project. Students were interviewed on their final projects. Transcription and analysis of these is yet to completed.

Additional instruments such as a *"Preparation of Future Learning*" [22] test designed to assess transfer of learning to a text-based programming language [11] are outside the scope of this paper. The Results section focuses on the post-test in which questions from the Israel national exam were used and additional questions akin to them designed for an online test. The open-ended summative assessments that included student-created games as well as interviews with students in keeping with the ideas of holistic assessment discussed earlier are not discussed in this paper.

4. RESULTS

Figure 3 shows the pre-post test scores (both averaged out of 100) in Studies 1 and 2. All the points are above the 45-degree line, indicating that all students had higher averages on the post-test than the pre-test. Statistical tests t-tests on the difference of the mean learning gains revealed that the learning gain in Study 2 was significantly higher than that in Study 1. Table 1 shows the test scores by gender, which suggests that girls in this sample performed significantly better than boys, although there were no significant differences by age or grade.

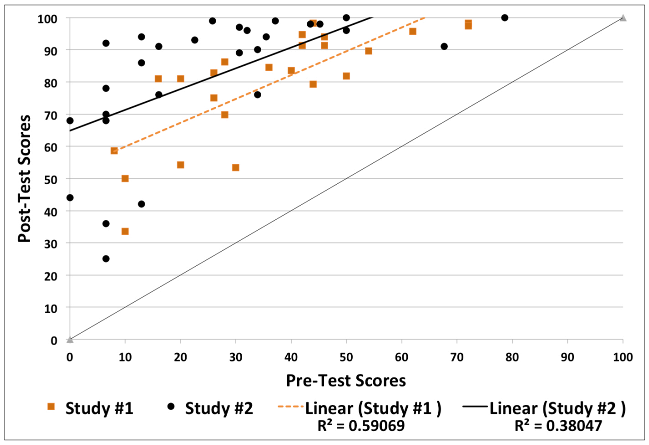


Figure 3: Pre-Post test average scores in Study #1 & Study #2

Table 1: Student Test Outcomes by Gender

	Mean (SE)		p-value
By Gender	Male (n = 40)	Female (n = 12)	
Pre-Test	29.9 (3.3)	38.4 (4.8)	0.20
Post-Test	77.1 (3.2)	89.7 (2.5)	0.04*

Note: p-values for the Pre-Test and Post-test come from a t-test of equality of means across samples with unequal variance. Given the non-normal distribution of the Post-test, the Mann-Whitney (Wilcoxon) rank sum test was also used and the p-value in that test was 0.07 for Post-test scores by gender.

Figure 5 shows student performance in the 6 questions containing 22 sub-questions that were re-used from the 2012 Israel national exam. It compares student performance for the 54 participants in our two studies with that of the 4082 students in Israel [28]. We found the difference in performance on all questions not statistically significant except in Question 9 in which our students performed significantly better than the Israel students scoring an average of almost 70% as compared to their 57%. We discuss this in more detail in Section 4.1. The Israel study report broke down the assessment measures according to **thinking skills** based on a modified version of Bloom's taxonomy as comprising Remembering & Understanding, Applying & Analyzing, and

Evaluating & Creating. The figures for the 2012 Israel Exam were 82%, 78% & 64% respectively for these three categories. Those corresponding figures for our students were 83%, 85% and 74%, however it is *important to note that a fair comparison cannot be made as the calculations in the former were made based on the whole exam comprising 9 questions; and ours are based only on the 6 (out of those 9) questions that we used.*

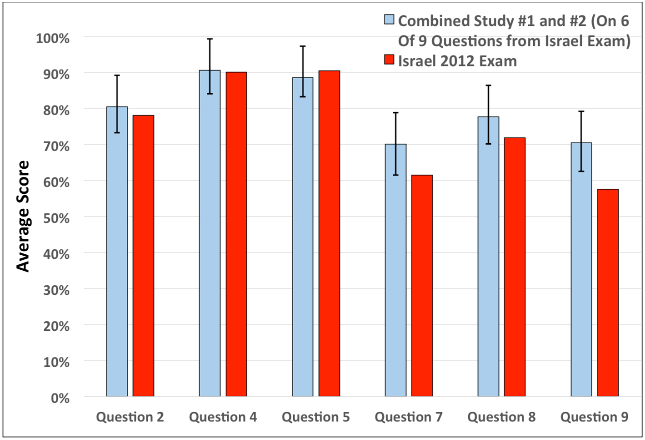


Figure 4: Comparison of Student Performance in Study #1 & #2 vs. 2012 Israel results (on 6 of 9 questions)

Additionally questions in the Israeli study were categorized according to **topics taught**, classified as serial execution (8%), conditionals (30%), Forever/Forever-If (12%) and For Loops (50%). The numbers in parentheses represent the percentage of the exam grade that was associated with that topic. The Israeli students' scores across the whole exam on the four topics was reported to be 87%, 85%, 89% & 64% respectively. We could not calculate corresponding figures for these as we were missing some questions, however the breakdown of *our entire exam* by topics taught is seen in Table 2 below.

Table 2: Post-Test Scores Breakdown by CS Topics Taught

	Mean (SD)
Overall Score	80.6 (19.2)
By CS Topic	
Vocabulary	72.9 (20.6)
Serial Execution	94.1 (17.6)
Conditionals	84.7 (19.6)
Loops	75.7 (24.1)

4.1 Discussion of Results

Based on results shown above, our students seemed to learn well for both Study 1 and 2 iterations of the 6-week FACT module. Based on student performance on the post-test in Study 1, we altered some strategies in Study 2- devoting more teaching time to certain topics, and a few different examples and programming assignments. This clearly helped as the student performance in the post-test in Study 2 improved.

Regarding the comparison of our students' performance following FACT on the same questions given to the students in Israel in 2012 following their national curriculum, it is worth noting the following distinctions in the research contexts. The Israel results were based on a sample of 4,082 7th grade students, while both of our research studies included both 7th and 8th grade students, comprising a total of 54 students. The Israel nationwide exam was given following a yearlong Introductory CS class. According to information provided by Zur Bargury to the lead author, the class was held for 2 hours a week for a school year, a total of 60 hours. The results shared here from our studies are from the post-test taken after roughly 24 hours over 6 weeks of learning with the FACT module. While we have demographic and other data, there is no such data available to us on the Israel students. In the Israel study, students taking this class were selected from within their schools as those "who excelled in their age group." In our studies, students were in the elective class based on stated interest or a counselor placing them in this elective; it was not seen as a part of the core curriculum or connected to official testing.

Despite these differences in the Israeli context from ours, most results were statistically no different. The only difference that warrants comment is that our students' performance in both Study 1 and Study 2 on Question 9 were significantly better than those of students in Israel. This was the only question that did not have multiple choice answers provided but required students to fill in 10 blanks in a Scratch script and involved the highest level of thinking, "Evaluating and Creating" in Bloom's taxonomy. Our students' success on this question may be due to a curricular focus on deeper understanding of concepts as well as practice in tracing existing code and reading/writing pseudo-code. Not having more details on the curriculum used in Israel, we are hesitant to comment on what may have affected the result in their case.

5. CONCLUSIONS & FUTURE WORK

Although student results on the post-test are encouraging overall, we hope to get a more holistic view of student learning, especially for children who did not perform as well on the test. To this end we are currently coding student interviews and grading final projects from Study 2. Preliminary results suggest that decontextualized assessment measures requiring reading abilities to understand written questions may not favor the English Language Learners in the student population. The projects and interview shows evidence of understanding of computational concepts even among low performers in the post-test, in addition to the obvious increased confidence and engagement levels when describing their own projects in contrast to discussing a question from the post-test, where they appear to get confused about some aspects of questions that appear ambiguous to them.

It is noteworthy that efforts to improve the instruction and Scratch assignments in Study 2 following Study 1 resulted in learning gains in Study 2 despite using an all-online version of FACT. We attribute the success of the FACT curriculum, especially on advanced questions like #9 of the Israel exam, to a good balance between explanation, demonstration of worked examples, use of pseudo-code, regular assessments that required children to trace (read) Scratch script and answer questions based on them, and several hands-on assignments in Scratch. However, future research is required to tease apart the productive conditions of the learning environment. Though unguided or minimally guided instructional approaches are popular among teachers employing easy-to-use environments like Scratch, we were guided by the argument that these approaches are less effective and less efficient than instructional approaches that place a strong emphasis on guidance of the student learning process [16]. A curriculum such as the one proposed in this research aims to help students see deeper structures in their computational artifacts and assess this learning via appropriate assessments.

A salient finding from our reuse of questions from the Israel exam is that student performance seems to be remarkably similar despite the distinctions between our research contexts, and the curriculum, location, student population as well as sample size taking the Israel National Exam. Perhaps more interesting is that both efforts are being used in part to hone a CS curriculum. Given that two completely unrelated curricula taught across the world to two very disparate groups of students presented such similar results suggests that the ease or difficulty that students face in learning certain computational concepts transcends teaching methods and materials, and are perhaps a function of age and cognitive maturity more than anything else.

A significant contribution of our research is the demonstration of the use of multiple forms of assessment in a structured introductory CS curriculum in a K-12 setting. Neither multiple-choice questions nor open-ended projects alone tell the whole story of student understanding. It would be unwise to ignore learner agency, motivation, creative expression and design thinking that students bring to projects of their own choosing [2]. This is especially critical when one of the stated goals of introducing CS is to inspire children to pursue this discipline and broaden the CS pipeline. However, it would be equally imprudent to not include objective measures that can be scaled and assess students' understanding of core computational concepts as well as associated skills such as debugging and code-tracing.

Perhaps the most noteworthy aspect of this effort is the reuse of assessment ideas and instruments from prior and ongoing efforts in different parts of the world *to build a cumulative knowledge base of a learning science for computational thinking*. This is especially pertinent as our individual nations move concurrently but separately towards a shared goal of building a computationally literate generation of learners. Leveraging the efforts of others and testing curricular instruments in new settings helps validate ideas and move the field forward.

6. REFERENCES

[1] Barron, B. 2004. Learning ecologies for technological fluency: Gender and experience differences. *Journal of Educational Computing Research*, 31(1), 1-36.

[2] Barron B. & Daring-Hammond, L. (2008). How can we teach for meaningful learning? In Daring-Hammond, L., Barron, B., Pearson, P. D., Schoenfeld, A. H., Stage, E. K., Zimmerman, T. D., Cervetti, G. N., & Tilson, J. L. 2008. *Powerful learning: What we know about teaching for understanding*. Jossey-Bass

[3] Biggs, J. B., & Collis, K. F. 1982. *Evaluating the quality of learning*. New York: Academic Press.

[4] Bornat, R. 1987. *Programming from first principles*. Prentice Hall International.

[5] Brennan, K., & Resnick, M. 2012. New frameworks for studying and assessing the development of computational thinking. *Paper presented at AERA 2012, Vancouver, Canada.*

[6] Cooper, S., Pérez, L. C., & Rainey, D. 2010. K-12 computational learning. *Communications of the ACM*, *53*(11), 27-29.

[7] Conley, D. T., & Darling-Hammond, L. (2013). Creating Systems of Assessment for Deeper Learning.

[8] Ericson, B., & McKlin, T. 2012. Effective and sustainable computing summer camps. *Proceedings of the 43rd ACM technical symposium on CS Education*, 289-294.

[9] Glass, A. L., & Sinha, N. 2013. Multiple-Choice Questioning Is an Efficient Instructional Methodology That May Be Widely Implemented in Academic Courses to Improve Exam Performance. *Current Directions in Psychological Science*, *22*(6), 471-477.

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

[11] Grover, S., Pea, R. & Cooper, S. (2014). Expansive Framing and Preparation for Future Learning in Middle-School Computer Science. In Proceedings of the 11th International Conference of the Learning Sciences (2014), Boulder, CO

[12] Ioannidou, A., Repenning, A., & Webb, D. C. 2009. AgentCubes: Incremental 3D end-user development. *Journal of Visual Languages & Computing*, *20*(4), 236-251.

[13] Lemke, J.L. (1990). *Talking science: Language, learning and values*. Westport, CT: Ablex Publishing.

[14] Lewis, C. M. (2011). Is pair programming more effective than other forms of collaboration for young students? *Computer Science Education*, *21*(2), 105-134.

[15] Lopez, M., Whalley, J., Robbins, P., & Lister, R. 2008. Relationships between reading, tracing and writing skills in introductory programming. *Proceedings of the 4th International Workshop on Computing Education Research*, 101-112.

[16] Mayer, R. E. (2004). Should there be a three-strikes rule against pure discovery learning?. *American Psychologist*, *59*(1), 14.

[17] Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M., 2010. Learning computer science concepts with Scratch. *Proceedings of the Sixth International Workshop on Computing Education Research (ICER '10)*, 69-76.

[18] Moskal, B., Lurie, D., & Cooper, S. 2004. Evaluating the effectiveness of a new instructional approach. *ACM SIGCSE Bulletin*, 36(1), 75-79.

[19] Parsons, D. & Haden, P. 2006. Parson's programming puzzles: a fun and effective learning tool for first programming courses. *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*, 157-163.

[20] Pellegrino, J. W., & Hilton, M. L. (Eds.). (2013). *Education for life and work: Developing transferable knowledge and skills in the 21st century*. National Academies Press.

[21] Piech, C., Sahami, M., Koller, D., Cooper, S. & Blikstein, P. 2012. Modeling how students learn to program. *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, 153-160.

[22] Schwartz, D. L. & Martin, T. (2004). Inventing to prepare for future learning: The hidden efficiency of encouraging original student production in statistics instruction. *Cognition and Instruction*, *22(2)*, 129-184.

[23] Scott, J. 2013. The royal society of Edinburgh/British computer society computer science exemplification project. *Proceedings of ITiCSE'13*, 313-315.

[24] Sweller, J., & Cooper, G. A. 1985. The use of worked examples as a substitute for problem solving in learning algebra. *Cognition and Instruction*, *2*(1), 59-89.

[25] Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. 2012. The Fairy Performance Assessment: Measuring Computational Thinking in Middle School. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, 215-220.

[26] Wing, J. 2006. Computational Thinking. *Communications of the ACM*, 49(3), 33-36.

[27] Zur Bargury, I. 2012. A new Curriculum for Junior-High in Computer Science. *Proceedings of ITiCSE'12*, 204-208, Haifa, Israel.

[28] Zur Bargury, I., Pârv, B. & Lanzberg, D. 2013. A Nationwide Exam as a Tool for Improving a New Curriculum. *Proceedings of ITiCSE'13*, 267-272. Canterbury, England, UK.