Concepts before coding: non-programming interactives to advance learning of introductory programming concepts in middle school

Article *in* Computer Science Education · February 2019

DOI: 10.1080/08993408.2019.1568955

Shuchi Grover
94 PUBLICATIONS **4,277** CITATIONS
SEE PROFILE

Nicholas Jackiw
SRI International
22 PUBLICATIONS **577** CITATIONS
SEE PROFILE

Patrik Lundh
SRI International
15 PUBLICATIONS **87** CITATIONS
SEE PROFILE

Concepts before coding: non-programming interactives to advance learning of introductory programming concepts in middle school

Shuchi Grover, Nicholas Jackiw & Patrik Lundh

To cite this article: Shuchi Grover, Nicholas Jackiw & Patrik Lundh (2019): Concepts before coding: non-programming interactives to advance learning of introductory programming concepts in middle school, Computer Science Education, DOI: 10.1080/08993408.2019.1568955

To link to this article: https://doi.org/10.1080/08993408.2019.1568955

Published online: 06 Feb 2019.

Submit your article to this journal

Article views: 13

View Crossmark data

COMPUTER SCIENCE EDUCATION
https://doi.org/10.1080/08993408.2019.1568955

ARTICLE

Concepts before coding: non-programming interactives to advance learning of introductory programming concepts in middle school

Shuchi Grover[a], Nicholas Jackiw[b] and Patrik Lundh[b]

[a]Looking Glass Ventures, Palo Alto, CA, USA; [b]SRI International, Menlo Park, CA, USA

ABSTRACT

Background and Context: Learners struggle with conceptual understanding of introductory programming concepts such as variables, expressions, and loops.

Objective: We examine whether and how designed activities for conceptual exploration support preliminary engagement with and learning of foundational and often hard-to-grasp programming concepts for students in grades 6–8.

Method: Drawing on principles from dynamic mathematics, we developed a suite of non-programming digital and unplugged activities embedded in a curriculum before students engage in Scratch block-based programming. We conducted empirical research in three middle school classrooms in diverse urban US schools and examined student performance through mixed qualitative and quantitative methods.

Findings: Learning gains were significant and not predicted by grade, gender or prior academic preparation. Free-choice projects of students showed statistically greater (correct) use of key concepts compared to those not in the study.

Implications: Our work demonstrates the promise of novel approaches such as interactive non-programming activities for deeper understanding of programming concepts.

ARTICLE HISTORY
Received 5 August 2018
Accepted 9 January 2019

KEYWORDS
Computer science education; CS concepts; introducing programming; novice programmer; curriculum design; K-12 CS education; variables; expressions; Boolean logic; abstraction

Introduction

Policy and educational leaders see computer science (CS) skills as necessary for all citizens, not just computer scientists, in order to build a strong STEM and computing pipeline and to develop future citizens with the problem-solving abilities needed to thrive and innovate in a world driven by computing and digital devices (The White House, 2016). Learning to program is a key ingredient of introductory CS curricula in K-12 classrooms. Learners today are typically introduced to programming in block-based programming environments designed to provide engaging features and syntactical supports for novice learners (Bau et al., 2017). However, programming is a complex activity

CONTACT Shuchi Grover shuchig@cs.stanford.edu Chief Learning Scientist, Looking Glass Ventures, 202 Sequoia Avenue, Palo Alto, CA 94306, USA

that novices of all ages find difficult regardless of the introductory programming environment (e.g. Robins, Rountree, & Rountree, 2003). Weak mental models and conceptual understandings prevent deeper learning of foundational CS concepts (Mayer, 2004; Robins et al., 2003). Franklin et al. (2017) and Grover, Pea, and Cooper (2015, 2016) report that students struggle more with certain introductory concepts than others in the context of programming. As middle school students learn to reason abstractly in the Piagetian formal operational stage, they need pedagogically robust experiences that provide a foundation for key concepts in CS and programming.

To address the need to expand teaching and learning of programming and CS, we explore new pedagogical approaches that support deeper conceptual understanding. Our research explores the impact of interactive, non-programming activities and technology-based microworlds on middle school students' conceptions of variables, expressions, loops and abstraction. We see a *rich and dynamic* understanding of variables as a cornerstone of students' encounters with all of these "VELA" concepts: variables (V) are the edifice of expressions (E); which in turn are critical linguistic drivers of iterative and looping (L) control structures; and variables are seen in CS education as the gateway to broader understandings of computational abstraction (A) and generalization (Figure 1). Recognizing the correlations between mathematics and programming understanding (as seen in Grover et al., 2015; Lewis & Shah, 2012; Shute, 1991), we draw inspiration from mathematics education research, where a pioneering curricular technology – the dynamic mathematics representation, has demonstrated positive effects in conceptual mathematics learning. For example, research on the use of SimCalc (an interactive mathematics software that employs a dynamic representational approach) in three large-scale studies with students in diverse demographic settings, established strong benefits to student conceptual understanding when dynamic representations are integrated into the plan for a curricular lesson sequence, and teachers have received related teacher professional development (Roschelle et al., 2010).

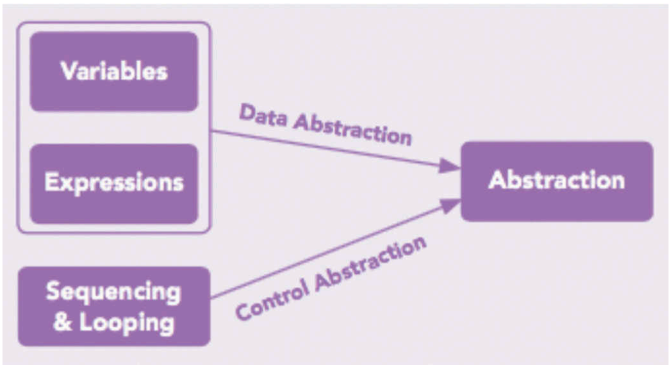


Figure 1. How abstraction builds on variables, expressions, and loops.

This paper first briefly describes five of the six activities of our VELA project: *Story Variables, Cats and Ladders, Graphical Looping, Three Switches*, and *Alarm Clock* that were designed, piloted and refined through a design-based research process (Wang & Hannafin, 2005). We then describe the empirical research conducted to examine the effectiveness of our intervention within an introductory CS course in three public middle school classrooms in a diverse urban school district in the US. We end with a discussion of our findings and plans for future work.

Literature review and theoretical framework guiding pedagogical designs

Our focus on the VELA concept trajectory is based on the large body of research devoted both to the difficulties of learning programming generally (for example, Du Boulay, 1986; Lahtinen, Ala-Mutka, & Järvinen, 2005; Pea & Kurland, 1984; Robins et al., 2003), and to students' specific difficulties with our focal concepts (for example, misconceptions around variables, or pervasive difficulties with variables in loops and abstraction). Today, block-based programming environments – such as Scratch, Alice, or Snap! – designed to address syntactical difficulties, are typical vehicles for introducing novice learners in K-12 to computational thinking and programming. Yet research shows that even though students find these visual environments easier to navigate, their conceptual difficulties pertaining to the semantics of coding still persist (Franklin et al., 2017; Grover & Basu, 2017; Grover et al., 2015).

Research documents difficulties with the concept of a variable in introductory programmer experiences, especially around initialization and misinterpretations of the common "box" or "drawer" didactic metaphor for storage (Du Boulay, 1986; Samurcay, 1989). Novices struggle with naïve notions about variable assignment (assuming variables can have multiple values at the same time); distinguishing between what goes inside a loop and what precedes or follows a loop (Du Boulay, 1986); and designing mathematical and logical expressions and naming variables (Gobil, Shukor, & Mohtar, 2009). While providing tools for inspecting variables as the program executes has been helpful in debugging code as well as understanding control and program state (Pears et al., 2007), such tools do not always help learners grasp how variables, expressions, and loops mutually interact to solve (or understand) a complex problem. Grover et al. (2015) found that seventh- and eighth-graders who had typical value-in-a-box variable visualization tools available (as in Scratch) still had trouble conceptualizing variables and how to use them in code, especially in constructing terminating conditions for loops involving variables and Boolean expressions (Grover et al., 2015). This echoes earlier research that problems pertaining to the understanding of conditionals are attributed to a lack of understanding of Boolean operators (Ebrahimi, 1994).

More recently, Grover and Basu (2017) also found that some students believe a variable must be a letter that is used as a short form for an unknown number (a carry-over from algebraic treatments of variable in math class), and students often assume multiple commands in a loop are repeated not as a unit but rather individually one after the other.

Past research also faults the possible lack of more pedagogic attention to – and engagement with – conceptual ideas for the problems novices face on their road to a robust understanding of programming (Pea & Kurland, 1984). Most K-12 programming classrooms are committed to student-centered activity and hands-on engagement with programming. But these activities are often centered on open-ended programming with minimal guidance, resulting in a lack of deeper or explicit engagement with, and understanding of, programming concepts and patterns (Mayer, 2004).

Pedagogical approach#1: constructivist engagement with concepts before coding

Our pedagogic approach in designing student-centered conceptually focused microworlds is influenced by *constructivism* with its twin emphases on knowledge as constructed (as opposed to transmitted) and on material engagement and manipulation as effective activity in such knowledge production (Ackerman, 2001).

We believe conceptually focused, constructivist experiences offer classrooms a path out of the minimally guided approaches critiqued above, through *designed* opportunities for students to acquaint themselves with *conceptual knowledge* through intentional student activity that also accommodates exploration. Our digital microworlds thus focus on *concepts rather than constructs*. This contrasts with much of the innovation that has occurred in the design of programming environments themselves (like Scratch, App Inventor, Blockly, PencilCode, Snap!, etc.), where students actually code and run programs. We design *non-programming* learning activities that allow students to explore and construct ideas for themselves, while also encountering powerful ideas embedded within the activities, thus balancing exploration and guidance – *before* they employ them in code (in a programming environment). This also draws on learning theory emphasizing that encountering concepts in multiple contexts helps learners to abstract the relevant features of concepts and hence promote deeper conceptual understanding (Bransford, Brown, & Cocking, 2000). Also, constructivist approaches to learning point to providing learners opportunities to actively construct knowledge by using their intuition and prior knowledge and refining their micro-theories as they interact with artifacts *before being provided explanations* (Schwartz & Bransford, 1998). Additionally, since contextualizing computational concepts makes it easier to learn programming (Papert, 1991), all our activity designs (as well as accompanying ideas for discussions and examples) are situated in relatable real-world contexts. We also strive to explicitly bridge non-programming activities and subsequent introductory programming activities in Scratch for mediating transfer between the two contexts (Grover et al., 2014).

Pedagogical approach#2: design principles drawn from dynamic geometry

Our technology environment design draws on the innovations of *dynamic geometry software* in mathematics education over the past 30 years, as typified by *The Geometer's Sketchpad* (Jackiw, 1991–2009) and *Cabri Géometre* (Laborde & Strässer, 1990) – two tools that have had global impact on mathematics instruction. Considerable research has explored the potential of dynamic geometry software in mathematical and curricular areas beyond its traditional home in high-school geometry (e.g. Schattschneider & King, 1997), often through connections predicated upon geometry's traditional status as the home of "mathematical insight" and upon dynamism's particularly effective relationship to mathematical ideas of the real continuum and the variable. We see dynamic representations as a potentially powerful vehicle for conceptual engagement among diverse learners in CS classrooms given its success in mathematics classrooms in large urban school districts (Roschelle et al., 2010).

From our synthesis of dynamic geometry software (across a spectrum of mathematical applications and variety of age levels), we developed design principles for the construction of introductory VELA activities and microworlds. These include the principles of: (1) *ubiquitous consistency*, whereby at any instant in time, a system of relationships is always entirely self-consistent: components of that system *always* accurately reflect their fundamental definitions or relationships to other components of the system, and there is no "uninitialized value"; (2) *dynamic variation*, whereby time is used as the principle axis of interrogation for both graphical and numerical parameters; varying these parameters offers tremendous potential for insight into the behavior of the system across inputs; (3) *graphical presentation*, whereby diagrams and visualizations are treated as primary representations and manipulatives in knowledge production, thus emphasizing the insightful value of the *illustrative* dimension of images and diagrams, as well as evolve them – through interactivity – into engines of knowledge *production* rather than just *presentation*; and, finally (4) *incremental abstraction*, where these first three principles contribute to learning trajectories in which abstraction is achieved incrementally through structured temporal variation. We believe these principles offer promise as an approach to balancing the tensions between an overly-material hands-on programming-first approach (that misses the opportunities for engaging with abstract concepts) and an overly theoretical or foundational approach to "essential definitions" (that misses the chance for experiential learning from the behavior of objects and systems). For more details on these principles and their manifestations in VELA activity designs, see Grover, Jackiw, and Lundh (n.d.).

VELA activities design

In this section, we describe five of the VELA activities (four digital and one unplugged) that we conceptualized and iteratively refined as part of the larger suite of both digital and non-digital activities for use in introductory CS classrooms to support early exploration and understanding of the VELA concepts in middle school (Table 1).

Wiggins & Tighe (2005)'s Understanding by Design framework guided our activity design process that began with an articulation of the precise target learning goals for our activities and companion curriculum (Table 2). Our digital activities were developed using a combination of Web Sketchpad and Javascript, and were deployed on the web (http://csforall.sri.com). Over a 15-month period, our multi-disciplinary research team comprising CS education and math education researchers, learning scientists, and a software developer iteratively designed and refined the activities based on inputs from teachers and students. Initial activity concepts were designed with classroom teachers from a large, urban school district in Western USA using a participatory design model.

Although each activity is intended to be modular and standalone so that it can be embedded in any introductory programming curriculum, we suggest a sequence. Our activities on *looping* and *variables* require no prior frame of reference, but activities on *expressions* draw on prior work with *variables. Abstraction*, as the capstone of our curricular materials, can be seen to draw on both looping activities' introduction of execution flow and "blocks" or "chunks" of functionality, as well as on the variable-and-expression activities' introduction of derived and compound data types. In the former context, we focus on *control abstraction* and in the latter on *data abstraction.*

Designed as a *short and preliminary exploration and introduction* to key ideas of foundational *concepts* (rather than introductory programming *constructs)* these activities do not attempt to deliver comprehensive treatments and are designed for one (or two) class periods. All digital activities begin with students exploring the basic phenomenology of the microworld. They are designed for exploration among student pairs interspersed with whole-class discussions.

Specifically, VELA activities introduce

Table 1. Suite of VELA activities [gray row indicates activity not discussed in this paper].

Name	Type	Introductory Computing Concept(s)
Graphical Looping	Digital	Repeating pattern, Looping, Modeling
Story Variables (lead-in to Cats and Ladders)	Unplugged	Variables (basic idea of variation, naming, datatypes)
Cats and Ladders	Digital	Variables, Arithmetic Expressions
Dice Conditionals	Unplugged	Conditionals, Boolean outcomes (TRUE/FALSE)
Three Toggles (VEA Microworld activity)	Digital	Boolean Logic, Boolean Expressions
Alarm Clock (VEA Microworld Activity)	Digital	Arithmetic and Boolean Expressions, Data Types, String Data, Modeling

Table 2. VELA learning goals.

Students will learn how programs are executed sequentially
Students will learn how simple loops work (fixed, predetermined number of repetitions)
Students will learn how to create different pathways in programs using conditional statements
Students will learn algorithmic flow of control–how instructions are executed in sequence even when there are loops, except that the set of instructions within a loop are repeated
Students will understand that in control structures (like loops and conditionals), a collection of an arbitrary number of statements can be declared to act as a single statement by grouping them
Students will understand what Initialization (in general and of variables, specifically) is and why it is needed
Students will understand that variables can only hold one value at a time
Students will learn how types define the set of values a variable can have, and the set of operators that can be used
Students will learn how variables are created, used, assigned values and updated
Students will learn how variable values change within loops
Student will learn what initialization is and why it is important
Students will learn how to use expressions to make new variables from old ones
Boolean variables, operators and expressions
Students will learn the idea of controlling loops and conditionals using Boolean conditions (that may or may not involve variables and expressions)
Boolean as a data type
Students will practice identifying and articulating patterns in real-world phenomena and problems, and abstracting them into structural components of a program (preconditions, repeating logic in a loop, any postcondition)
Students will learn how variables are an abstraction or representation of data in the program and the real world
Students will learn the importance of planning before programming
Students will learn the need for breaking down problems into smaller manageable tasks
Students will learn computational solutions are abstractions; and that these abstractions can be represented in different ways

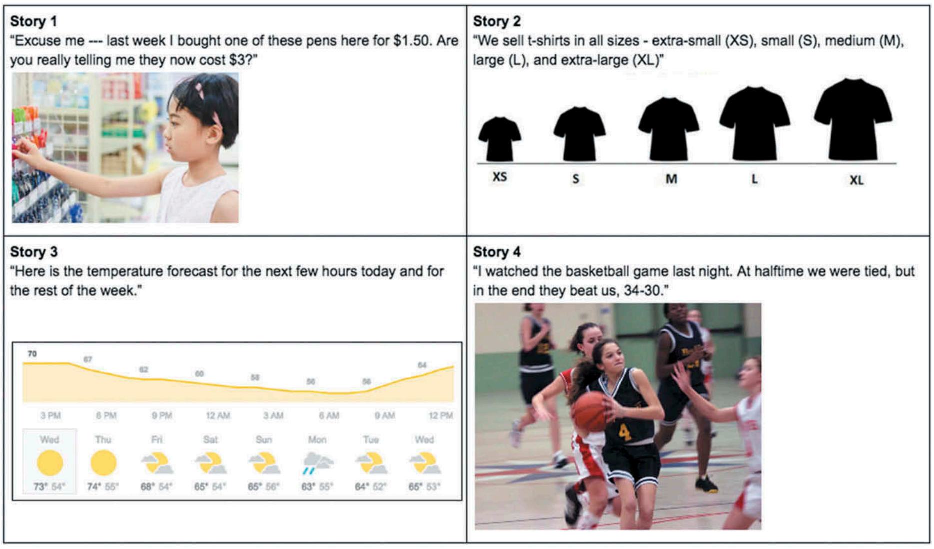
- *Variables* as named quantities that can change value
- *Expressions* as defining new changing values by applying value-appropriate operations to existing variables
- *Looping* as repetitions of some identifiable "repeating unit"
- *Abstraction* as the process of giving a name to a specific collection of details as a way of referencing its purpose without quoting or enumerating its detail.

We describe the activities and lesson sequence in brief in the remainder of this section.

Variables and arithmetic expressions: Story Variables (unplugged) and Cats and Ladders (digital)

The *Cats and Ladders* activity engages students with the concept of variables. It follows an unplugged (off-line) activity, *Story Variables*. In *Story Variables* students work collaboratively in pairs to investigate a series of short "stories" containing *quantities that vary* (Figures 2 and 3)). For example, "Excuse me – last week I bought one of these pens here for $1.50. Are you really telling me they now cost $3?"; "I watched the basketball game last night. At halftime we were tied, but in the end, they beat us 94–90". Through discussions, students come up with a definition of “variable,” practice identifying and naming variables meaningfully, and analyze a variable’s changing values to determine its specific types and expected ranges. Students articulate their own real-world scenarios that involve “variables”. As a final activity, they watch a video clip of a video game (e.g. Pacman) and list the different variables they observe, their values in the course of the video clip, and the logical ranges of values for each.

Cats and Ladders formalizes this understanding of variables as *named quantities with specific values that can change*. In the *Cats and Ladders* digital activity (Figure 4),

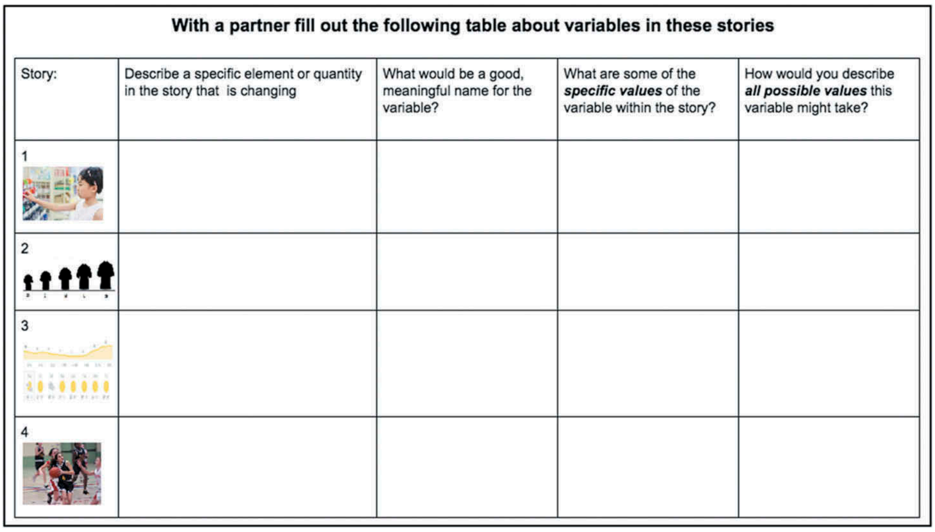

Story 1
“Excuse me --- last week I bought one of these pens here for $1.50. Are you really telling me they now cost $3?”

Story 2
“We sell t-shirts in all sizes - extra-small (XS), small (S), medium (M), large (L), and extra-large (XL)”

Story 3
“Here is the temperature forecast for the next few hours today and for the rest of the week.”

Story 4
“I watched the basketball game last night. At halftime we were tied, but in the end they beat us, 34-30.”

Figure 2. Short everyday narratives in “story variables”.


With a partner fill out the following table about variables in these stories

Story:	Describe a specific element or quantity in the story that is changing	What would be a good, meaningful name for the variable?	What are some of the ***specific values*** of the variable within the story?	How would you describe ***all possible values*** this variable might take?
1				
2				
3				
4				

Figure 3. Identifying variables, meaningful naming, and values in “story variables”.

students rescue distraught cats from the various floors of buildings by determining the length of the ladder required to reach them. The activity is divided into stages. Starting with an exploration of the basic behavior of the microworld, new stages are "unlocked" as the student proceeds through the activities. Students move from working with one (blue) ladder, to two stacked ladders (one orange and one blue) when the blue ladder height is insufficient to rescue cats on higher floors. Learners discuss appropriate names for variables (e.g. "height" or "LadderHeight" is not sufficiently discriminating for the two ladders). They also discuss the range of possible values (often determined by context), and that different variables may naturally reference different (data) types. Finally, they engage in abstraction through a preliminary exploration of arithmetic expressions and that new variables can be synthesized from existing ones (e.g. *TotalLadderHeight* = *BlueLadderHeight* + *OrangeLadderHeight*).

The game's final stage includes multiple buildings with alphabetic labels (Figure 4(d)) aimed at helping students interact with a two-dimensional coordinate system. In comparing this third variable – whose values are letters, rather than numbers – to the other two, students encounter an early example of non-numeric and compound data types. From here they discuss possible further *generalizations* of the scenario.

Looping (repetition): Graphical Looping (digital)

The *Graphical Looping* activity sequence (Figure 5) introduces students to iterated repetitions of a block of actions within a sequence of events, and develops the idea that we efficiently express such a flow of events in terms of a more *compact specification* of that repetition. Students engage with the idea of action sequences that occur before and/or after a repeating chunk of actions ("repeating unit"). *Graphical Looping* uses comic panels as a proxy for

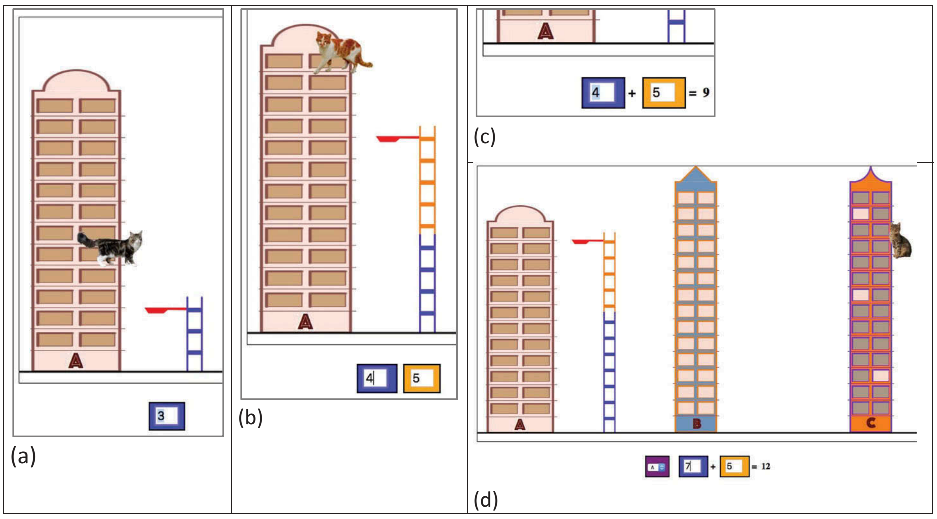


Figure 4. Screenshots of cats and ladders activity.

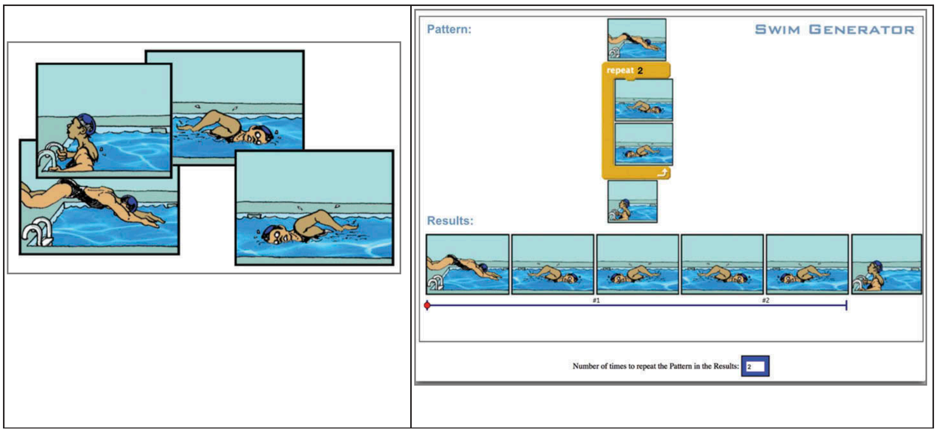


Figure 5. Screenshots of *graphical looping* activity.

source code in a pre-programming context. Comic strips are *atextual* (and thus do not disadvantage ELL students) and contain a formal, and block-structured, grammar for describing action sequences familiar to students.

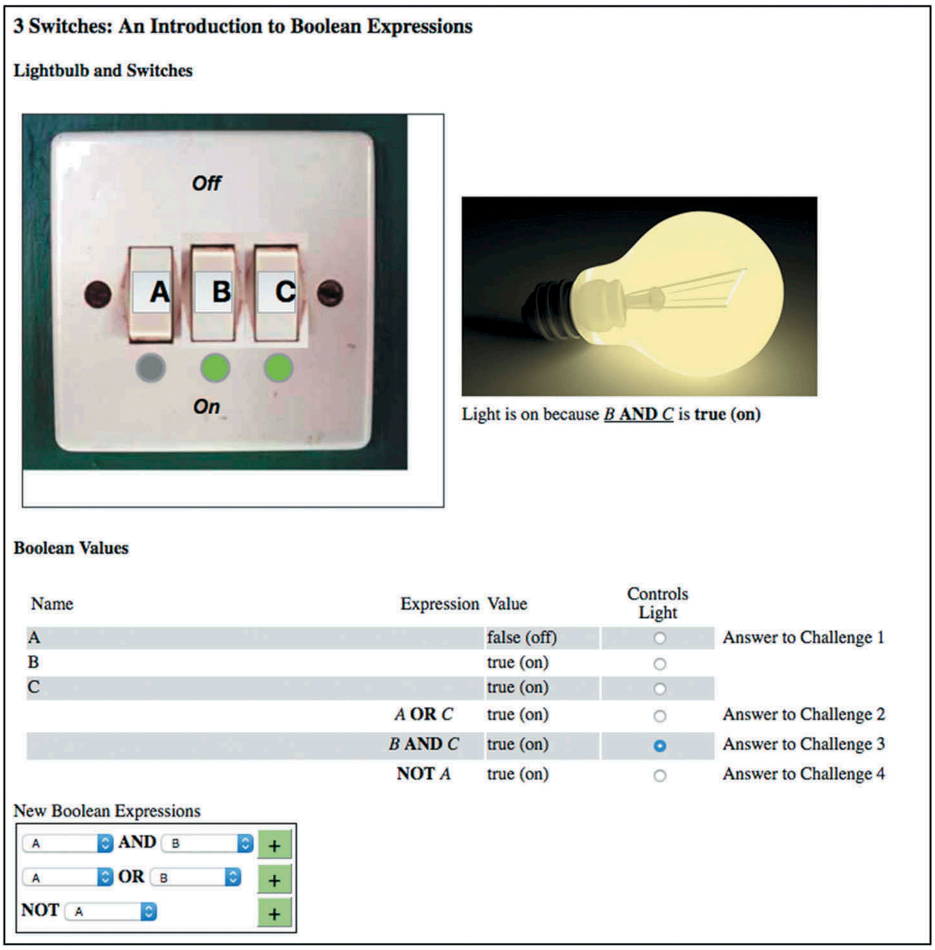
First, in *Swimming Pool Story Arrangement*, students arrange comic panels to tell a "logical story". Then, they proceed *From Arranging to Generating* – thinking about how panels to the story could describe longer swims and identify the "inner story" or "repeating unit". Students answer questions such as: What happens when the inner story is repeated zero times in the results? Once? If the repeat count is 10, how many pool lengths will the swimmer swim? How many panels will be in the result? How might the total length swum increase and the swimmer's energy decrease with each lap? These ideas are later revisited as examples in bridging VELA activities to Scratch.

Boolean operators and expressions, and abstraction: Three Switches (digital) and Alarm Clock (digital)

The last two digital activities in the VELA sequence introduce learners to Boolean operators and expressions, and, finally, abstraction through the concrete trope of *naming* a complex phenomenon in order to hide its details and complexity behind a simple name. Although distinct activities, both *Three Switches* and *Alarm Clock* are built on the same underlying expression–authoring microworld. In keeping with the VELA design philosophy, both activities are situated in real-world scenarios.

In *Three Switches* (Figure 6), the goal is twofold: (1) to explore the basic mechanics of the expression-authoring features of the microworld, and (2) to provide learners with a first introduction to Boolean values, operators, and expressions. Students explore how Boolean operators (AND, OR, NOT) work, and what the resulting expressions evaluate to using an example of three switches that toggle between two states (ON and OFF/TRUE and FALSE). They start with a series of four exploratory "challenges" that explain the types of Boolean operations. In Challenge #1, students discover through exploration that the bulb is controlled only by switch A. When A is on, the bulb is "on". In Challenge #2 Students find out that the bulb turns on when either Switch A is on, or Switch C is on, or both switches A and C are on. (Switch B does not control the bulb). In Challenge #3, the bulb turns on only when both switches B and C are on; and in Challenge #4, when Switch A is on, the bulb is off; and when Switch A is off, the bulb is on. The next step of the activity reveals to students that the *expressions* that control the bulb in challenges 1–4 are: A, A OR C, B AND C, and NOT A, respectively. Students then fill out truth tables for expressions A OR C, B AND C, and NOT A, and explain the behavior of the OR, AND, and NOT operators in words.

Students subsequently extend the ideas to create and evaluate Boolean expressions that use these operators either as simple expressions or



3 Switches: An Introduction to Boolean Expressions

Lightbulb and Switches

Light is on because *B* **AND** *C* is **true (on)**

Boolean Values

Name	Expression	Value	Controls Light	
A		false (off)	○	Answer to Challenge 1
B		true (on)	○	
C		true (on)	○	
	A **OR** *C*	true (on)	○	Answer to Challenge 2
	B **AND** *C*	true (on)	●	Answer to Challenge 3
	NOT *A*	true (on)	○	Answer to Challenge 4

New Boolean Expressions

A **AND** B +
A **OR** B +
NOT A +

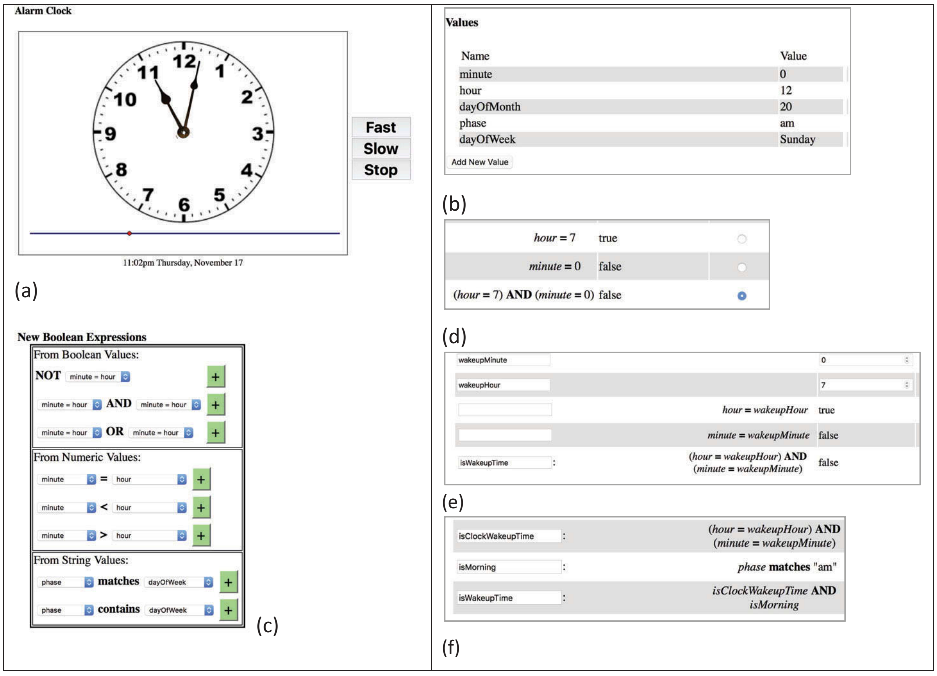
Figure 6. Screenshots of *three switches* activity.

compound expressions that combine one or more operators. The activity ends with an exploration of connections between variables and Boolean expressions in real-world scenarios through discussions that involve articulating the outcome variable, the controlling variables, the Boolean operators used, and the Boolean expression. For example –

(1) Travis' parents say that he can go out and play with his friends if he has done both – finished his homework and made his bed in the morning
(2) Maya will go on a hike if it is not raining.
(3) Keisha can watch a movie if she finishes either her math homework or her computer science project.
(4) A car gives a warning beep if –
    - Driver's seat-belt is not locked
    - Passenger's seat-belt is not locked
    - Car is being driven

In the culminating VELA activity, *Alarm Clock* (Figure 7), students construct a family of arithmetic and logical expressions using a variety of data types and starting from a variety of variables that characterize conditions under which an alarm clock should ring to wake them for school (when *IsAlarmTime* is True). Beginning, again, with phenomenological investigation, work in *Alarm Clock* is conducted in an expression-building microworld, in which a dynamic simulation – a graphical alarm clock – may be probed (analyzed, measured, mathematically modeled) by interrogating the values of several key variables, and by combining them with various operators into dynamic expressions. By assigning names to these new expressions, and by incorporating those new names (rather than the full symbolic expressions defining them) into new expressions that reference those intermediate results, students develop a hierarchy of abstractions to manage the burgeoning symbolic and conceptual complexity of a family of equations expressed only in terms of foundational variables.

Through this iterative revisiting, the modeling context is open to a variety of definitions of success depending on the interests and ability of the students. Some go on to aspire to only be woken for schools on *weekdays*, and extend their expressions further. In such a model, an expression such as (*isTimeForSchool AND isSchoolDay)* becomes a highly abstract proposition, encoding (and hiding) a non-trivial amount of decision-making involving times, clock mechanics, and day-calendars.



Values

Name	Value
minute	0
hour	12
dayOfMonth	20
phase	am
dayOfWeek	Sunday

hour = 7	true	○
minute = 0	false	○
(*hour* = 7) **AND** (*minute* = 0)	false	●

wakeupMinute		0
wakeupHour		7
	hour = *wakeupHour*	true
	minute = *wakeupMinute*	false
isWakeupTime :	(*hour* = *wakeupHour*) **AND** (*minute* = *wakeupMinute*)	false

isClockWakeupTime :	(*hour* = *wakeupHour*) **AND** (*minute* = *wakeupMinute*)
isMorning :	*phase* **matches** "am"
isWakeupTime :	*isClockWakeupTime* **AND** *isMorning*

Figure 7. Screenshots of alarm clock activity.

Empirical study: methods, data measures and analyses

Organizing the VELA curricular intervention

Working with classroom teachers, the VELA activities were incorporated into a 20-day intervention (Table 3) to be embedded within an introductory programming middle school curriculum. The VELA intervention comprised the following –

VELA activities

These included six VELA activities (the five described above and *Dice Conditionals*, an unplugged dice game activity to introduce students to true/false logic and conditional thinking, described in Grover, Lundh, & Jackiw, 2019), along with accompanying teacher lesson plans, student worksheets (to be completed in pairs in conjunction with the activities), and "review sheets" for formative assessment.

Scratch lessons

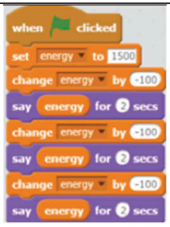
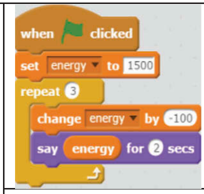
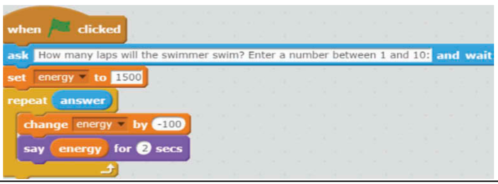
In addition to the VELA activities, we designed lesson plans to introduce students to basic Scratch constructs and programming concepts. The Scratch lesson plans also made connections to relevant ideas in VELA activities in an effort to mediate transfer. For example, the Scratch activities related to loops with variables made explicit connections to *Graphical Looping*. Learners are introduced to the idea of

Table 3. 20-day VELA intervention to be embedded in an introductory CS class.

	VELA activities	Descriptions
Day-0	Pre-assessment and Pre-survey	
Day-1	Graphical Looping (Digital Activity) – Swimming comic strip and other real world examples	Identifying repeating patterns and conceptualizing before/during/after of loops
Day-2	Simple Loops (Scratch)	Scaffolded and open-ended Scratch activities using simple “Repeat”, Drawing in Scratch, basic shapes.
Day-3	Nested Loops – Flowers and music (Scratch)	Scaffolded and open-ended Scratch activities using nested ‘Repeat’s
Day-4	Simple Conditionals – (Scratch)	Scratch activities (sensing and conditional in Maze activities)
Day-5	Story Variables (Unplugged Activity)	Introducing idea of changing quantities in the real world and giving them meaningful names
Day-6	Cats and Ladders (Digital Activity)	Naming variables and creating expressions in non-programming context.
Day-7	Creating and Updating Variables (Scratch)	Creating variables in Scratch; understanding how to initialize and update values
Day-8	Unplugged “Dice” game	Arithmetic epressions and conditionals based on epressions
Day-9	Conditionals with Arithmetic Expressions (Scratch)	Scaffolded and open-ended Scratch activities using variables, expressions and conditionals.
Day-10	Loops with Varialbles (Scratch)	Scaffolded and open-ended Scratch activities using variables, and loops
Day-11	Loops, Variables and Conditionals (Scratch)	Scaffolded and open-ended Scratch activities using variables, loops and conditionals
Day-12	VEA 3 Switches (Digital Modelling Activities)	Booleans, boolean operators and abstractions
Day-13	Games with Booleans (Scratch)	Scaffolded and open-ended Scratch activities using boolean logic
Day-14	VEA Alarm Clock (Digital Modelling Activities)	Modelling real world Alarm Clock situtions using variables, arithmetic and logical expressions and abstractions.
Day-15	Games using Repeat-Unitls (Scratch)	Scaffolded and open-ended Scratch activities using Repeat-Unitls loops, variables, expressions and conditionals.
Days-16-20	Games in Scratch	Make games more exciting by using all concepts learned
Day-21	Post-assessment and post-survey	

using a “Repeat” block to swim three laps, change the (swimmer’s) *Energy* value, and also “watch” this value decrease (using the “Say” block) with each lap (or iteration of the loop). They then progressed to the idea of a generalized solution where the number of laps was based on an input from the user to be used as a variable in the Repeat block (Figure 8).

The bulk of Scratch programming involved hands-on work in Scratch – completing partially coded programs using the relevant construct being introduced, creating complete programs to achieve a goal, and/or open-ended programs of choice. We also devoted attention to the construction and use of Boolean expressions and their use in conditional statements and the “Repeat Until” loop block – a non-deterministic loop construct that makes the loop repeat until some criterion is met. It requires the construction of a Boolean expression to determine when it ends, and the expression could use a variable that gets updated in the loop. This is conceptually more sophisticated than the simpler (and more commonly taught) form of looping taught in

	 
(a) Rewrite this code snippet using **a simple loop**. The swimmer starts with an energy of 1500 calories. You should "say" how many calories are left at each iteration of the loop.	(b) Now, you do not know beforehand how many laps the swimmer will swim. Open the starter program. You will see code (below) that asks the user to enter how many laps the swimmer will swim (a number between 1 and 10). Complete the program so that based on the number the user enters, the program will show how the swimmer's energy decreases. Show how many calories are left at each iteration of the loop.

```
when [flag] clicked
set energy to 1500
change energy by -100
say energy for 2 secs
change energy by -100
say energy for 2 secs
change energy by -100
say energy for 2 secs
```

```
when [flag] clicked
set energy to 1500
repeat 3
    change energy by -100
    say energy for 2 secs
```

```
when [flag] clicked
ask How many laps will the swimmer swim? Enter a number between 1 and 10: and wait
set energy to 1500
repeat answer
    change energy by -100
    say energy for 2 secs
```

Figure 8. Example of programming activities in the scratch that bridge to VELA.

Scratch – "Forever" loops with an IF-block conditional check within them – that is at odds with accepted practice of CS (Meerbaum-Salant, Armoni, & Ben-Ari, 2011). As with VELA activities, worksheets guided students work during Scratch programming.

Formative assessments

Every VELA activity and Scratch programming lesson included review questions to be completed by students individually. These extended students' thinking beyond the examples encountered in the lesson, provided feedback to teachers on student understanding, and informed subsequent whole-class reviews and discussions.

Pre-post assessment

We also designed, piloted and developed an assessment instrument using Evidence-Centered Design (Mislevy, Almond, & Lukas, 2003), a principled assessment design framework, to be used as a pre-post measure. The targeted skills were drawn from VELA learning goals as well as the K-12 CS framework (k12cs.org), and CSTA CS standards (2017, draft version at the time) so that *the instrument could be usable more broadly for assessing introductory programming skills in middle school CS*. The assessment was refined based on data from pilots in middle school classrooms teaching introductory Scratch programming (~200 students over 2 pilots), teacher feedback, student cognitive think-alouds and language simplification by an English Language Learner (ELL) expert. The 9 items, with 29 individually scorable sub-items, were a combination of multiple-choice and open response types; and used Scratch code snippets or non-programming, real-world scenarios. One sample question is shown in the Appendix. The entire assessment, along with difficulty, reliability and validity analyses, is described in Grover (2019).

Classroom research

We partnered with a large urban school district in Western USA for this research. IRB permissions were sought from teachers and students (and parents of students) involved in this research.

Beta versions of the VELA activities were piloted in controlled settings at three different time points before they were used in a classroom research study – (1) within our research lab during early prototyping, (2) at a local summer camp in Northern California in 1–1 and small group sessions with six middle school children and (3) with 16 students with varied levels of prior math preparation from two different middle schools in our partner district. Data from these pilots helped refine the design of the activities before the full classroom intervention.

The VELA curricular intervention was implemented in the partner district by three middle school CS teachers in sixth, seventh and eighth grades (each in a different school) with a total of 74 students. (Three students were dropped from data analyses due to missing permissions or pre-posttest data for a final *N* = 71 students). While the seventh grade was in a high-performing school, the other two schools were low performing. Grade 7 also had statistically higher prior math and ELA scores than Grades 6 and 8. Table 4–5 describes the sample by gender, grade and ethnicity, and the school descriptions.

Before implementing the VELA curriculum, the teachers participated in two days of professional development. Teachers learned about the VELA rationale and the principles guiding the curriculum design. They participated in role-playing during curriculum enactment, engaged in hands-on activities to experience using the curriculum and reviewed lesson plans. They were encouraged to modify any lesson plan as well as tweak the pace and timeline of the curriculum, if necessary.

We conducted mixed-method (including quasi-experimental) research to determine (a) *whether and how well students learned through the VELA curricular intervention, and (b) how student learning was linked to students' demographic factors (such as prior math and English preparation, gender, ethnicity and grade level)*. These research questions were also aimed at understanding the

Table 4. Student sample and demographics.

	Grade 6	Grade 7	Grade 8
Male	17	15	11
Female	9	16	3
Asian	17	22	7
American Indian	1	0	0
White	1	2	0
Declined to State	0	5	0
Multiracial	2	1	1
Filipino	0	1	3
Hispanic/Latino	4	0	3
African American	2	0	0

Table 5. School (grade) characteristics.

	School 1 (Grade 6)	School 2 (Grade 7)	School 3 (Grade 8)
Low SES	85%	40%	87%
Math Proficiency	55%	79%	34%
English Proficiency	54%	80%	41%
Science Proficiency	62%	84%	34%
Disabilities	12.8%	3.5%	15%
Facility Condition	Good	Exemplary	Good

design modification needs of the VELA activities as guided by DBR's iterative approach. *A third, broader thread of inquiry* ***not covered in this paper*** *involves conducting and analyzing case studies of the classroom activity system (including student and teacher cases) in order to address the question: What factors and conditions should be highlighted in a theory of action for engaging diverse middle school CS classrooms, including factors related to students, to teachers, to resources and to the environment?*

Data sources and analysis

We collected qualitative and quantitative data to inform our analyses of student learning and experiences with VELA activities. Data measures included:

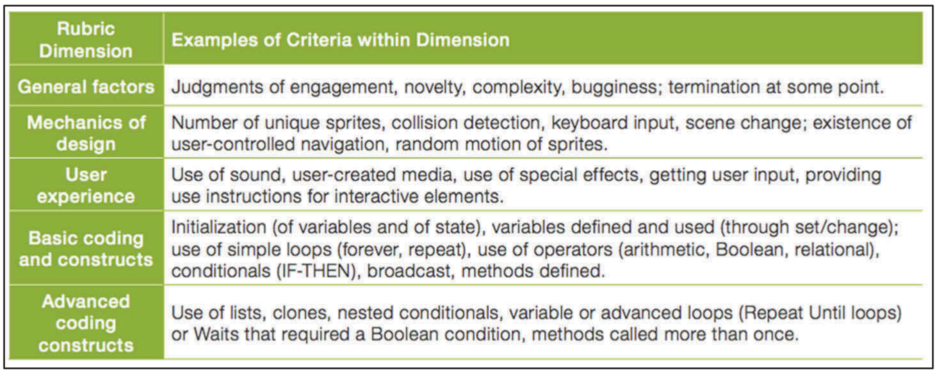
- *Pre-post assessment*: a paper–pencil assessment completed in one class period.
- *Student Interviews*: of four students from each class conducted twice – during and after the intervention.
- *Individual Teacher Interviews and Focus Group sessions*: with the three teachers piloting the VELA curricular intervention (for us to receive feedback on the VELA activities)
- *Post-VELA Surveys*: with Likert-scale items addressing engagement with VELA activities and understanding of VELA concepts.
- *Prior Math and English Scores*: from the standardized state tests were provided by the district for each student in our sample.
- *Final, free-choice (open-ended) projects in Scratch (VELA Students)*: from the three classrooms, done (individually or in pairs) as a culminating activity of the CS course in which the VELA intervention was embedded. As this was completed at the end of the 9-week CS course (well after the end of the VELA intervention), we received a total of only 54 projects across the three grades.
- *Final, free-choice (open-ended) projects in Scratch (non-VELA Students)*: as a comparison, we also examined 60 final Scratch projects provided by 20 teachers from middle school classrooms across the same district that were not part of this study. These students completed the middle school CS

course adapted from Harvey Mudd's *MyCS* curriculum (Schofield, Erlinger, & Dodds, 2014) as part of the district's ongoing CS expansion efforts. These projects were randomly selected by teachers and similarly done by individuals or pairs of students as a culminating activity.

This paper reports on analyses of data from the pre-posttest, post-VELA surveys and interviews, and mixed-methods analyses of students' open-ended Scratch projects. We conducted *paired t-tests* of pre- and post-test scores, statistical comparisons and correlational analyses (Pearson correlations, regressions) of posttest scores and learning gain (*learning gain = posttest – pretest score*) by gender, grade, and prior math and English scores.

Scoring of open-ended projects

We scored students' open-ended free-choice Scratch projects using an extensive rubric from Grover, Basu, and Schank (2018) publicly available at https://goo.gl/SEpY51. The rubric, adapted from Grover (2014), was created mainly for evaluating Scratch projects rather than providing students a project score based on a traditional student-facing rubric. It also included assessments of program complexity, creativity and correctness. Figure 9 shows the main dimensions of the Scratch rubric – "general factors", "mechanics of design", "user experience", "basic coding constructs", and "advanced coding constructs", and what each of them measures. The separation of "user experience" and "design mechanics" is useful as it separates design mechanics ("design pattern"-like code) such as using random motion, collision detection, and keyboard input from the elements of creativity students use in Scratch as part of engagement and user experience. "General Factors" assess overall proficiency such as novelty, correctness (or bugginess), and complexity. "Advanced Coding Constructs" includes the types of constructs that suggest more sophisticated program structures, for example, "Repeat-Until" loops, Wait



Rubric Dimension	Examples of Criteria within Dimension
General factors	Judgments of engagement, novelty, complexity, bugginess; termination at some point.
Mechanics of design	Number of unique sprites, collision detection, keyboard input, scene change; existence of user-controlled navigation, random motion of sprites.
User experience	Use of sound, user-created media, use of special effects, getting user input, providing use instructions for interactive elements.
Basic coding and constructs	Initialization (of variables and of state), variables defined and used (through set/change); use of simple loops (forever, repeat), use of operators (arithmetic, Boolean, relational), conditionals (IF-THEN), broadcast, methods defined.
Advanced coding constructs	Use of lists, clones, nested conditionals, variable or advanced loops (Repeat Until loops) or Waits that required a Boolean condition, methods called more than once.

Figure 9. Rubric dimensions for analysis of scratch projects, and examples of criteria within each rubric dimension.

Until, methods, and cloning. These two category scores are weighted higher than other categories in the overall score as they are reflective of higher program sophistication (and by extension, deeper conceptual understanding). Additionally, since frequency of a construct is not necessarily indicative of program sophistication, the rubric-based coding/scoring was based on one of the following: (1) determinations about the existence or not (1/0) of a given criterion (e.g. Does the program terminate at some point? Are instructions provided for interactive elements? Do variables have meaningful names?), (2) count of the number of times a construct was ***correctly*** used (most were capped at 10 to account since code segments in Scratch are often repetitive), (3) score along a 3-point scale (e.g. many bugs, few bugs, no bugs). In projects that included several sprites with identical or near-identical code, code *associated with only one sprite was considered*.

We ran *two-sample t-tests* to compare these scores and sub-scores to those in the comparison dataset (both sets were scored using the same rubric) along the rubric dimensions. We also specifically compared the use (frequency and correctness) of variables, Boolean and relational operators and various types of loops (especially "Repeat-Until" loops). We analyzed the post-VELA surveys as well as student and teacher interviews for quantitative and narrative feedback on the VELA activities.

Empirical results and discussion

We report on the quantitative analyses related to the pre-post assessment and qualitative scoring of final Scratch projects.

Pre-to-post scores

This section presents salient findings from statistical analyses of posttest scores and learning gain.

All three classrooms showed statistically significant gains on average on the pre-post assessment (See Table 6). Grade 7 students had a statistically higher average pre-score than Grade 6 ($t(55) = -5.83$); $p < 0.001$). Grade 8 students also had a statistically higher average pre-score than Grade 6 ($t(38) = -2.59$); $p = 0.01$). However, there was no statistical difference between Grades 7 and 8 average pre-scores ($t(43) = 1.91$; $p = 0.06$). Grade 6 had a statistically lower post-score than both Grade 7 ($t(55) = -5.72$); $p < 0.001$) and Grade 8

Table 6. Student pre-post results on VELA assessment.

	pre Mean (SD)	post Mean (SD)	*t*	*p*-value	Cohen's *d*
Grade 6 (*n* = 26)	41.3 (17.6)	53.2 (20.7)	−3.55	0.00	0.65
Grade 7 (*n* = 31)	65.5 (13.8)	80.0 (14.0)	−6.13	0.00	1.02
Grade 8 (*n* = 14)	56.3 (17.4)	68.5 (18.9)	−4.27	0.00	0.67

(*t*(38) = −2.28; *p* = 0.03). Grade 8 also had a statistically lower average post-score than Grade 7 (*t*(43) = 2.22; *p* = 0.03).

Although all our analyses presented in this paper include all items of the pre-post assessment, it is worth pointing out that one item on the pre-posttest (Item#2) was found to be problematic as the language of the question was ambiguous. This occurred due to a last-minute change to the question which resulted in the item being used for the VELA intervention without being tested in classrooms prior to its use. The pre- and post-scores for the item for all three grades were very low, and the difficulty level was 0.35 (the highest difficulty among all the items in the assessment). Item#2 scores normalized as a percentage (out of 100) were as follows: Grade 6 pre-score: *M* = 16.2, *SD* = 18.8 and post-score: *M* = 20.8, *SD* = 29.1; Grade 7 pre-score: *M* = 44.5, *SD* = 30.0 and post-score: *M* = 46.5, *SD* = 35.2; Grade 8 pre-score: *M* = 34.3, *SD* = 28.7 and post-score: *M* = 40.0, *SD* = 40.0. The item has since undergone revisions and is currently being field-tested with various student populations studying introductory programming in Scratch.

There was no statistically significant difference in average pre-score between females (*M* = 58.7; *SD* = 18.3) and males (*M* = 52.3; *SD* = 19.5); *t* (69) = 1.38; *p* = 0.17. Though females scored higher than males on average in the posttest, there was no statistically significant difference in the average post-score either between females (*M* = 73.5; *SD* = 20.3) and males (*M* = 64.0; *SD* = 21.0); *t*(69) = 1.88; *p* = 0.06. Similarly, average learning gains were not statistically different for females (*M* = 14.8; *SD* = 12.3) and males (*M* = 11.7; *SD* = 15.1); *t*(69) = 0.91; *p* = 0.37.

Prior math score was a significant predictor of pre-score; English was not (Table 7). However, when controlling for pre-score, *neither math nor ELA were significant predictors of post-score* (Table 8). Since the distribution of students by ethnicity was uneven and sample sizes very small for most groups, we could not perform any meaningful analyses by ethnicity. *Learning gain* was not statistically different by or predicted by gender (as mentioned above), grade (*F*(2, 68) = 0.20; *p* = 0.12), or prior math or English score (Table 9) – i.e. the

Table 7. Regression to test predictors of pre-score (*N* = 71; *R2* = 0.51).

	Coeff.	Std. Err.	t	P > t
Prior English Score	2.58	2.79	0.93	0.36
Prior Math Score	9.02	2.49	3.63	0.00
Intercept	20.17	6.49	3.11	0.00

Table 8. Regression to test predictors of post-score (*N* = 71; *R2* = 0.66).

	Coeff.	Std. Err.	t	P > t
Pre-score	0.57	0.10	5.42	0.00
Prior English Score	4.22	2.34	1.80	0.08
Prior Math Score	3.44	2.33	1.48	0.14
Intercept	13.73	5.02	2.74	0.01

Table 9. Regression to test predictors of learning gain (difference from pre-score to post-score) (*N* = 71; *R2* = 0.04).

	Coeff.	Std. Err.	t	P > t
Prior English Score	3.06	2.59	1.18	0.24
Prior Math Score	−0.44	2.37	−0.19	0.85
Intercept	5.27	5.09	1.04	0.30

intervention showed no statistically significant difference in *learning gains* in students by gender or prior academic preparation, and all grades showed statistically similar learning gains.

We find this last finding to be very significant in that it suggests that learning gains were not linked to specific grades or reported student characteristics. It demonstrates the promise of the VELA approach and curriculum to reach all students regardless of prior preparation. Also, prior math scores losing significance as predictor of post-score when controlling for pre-score is a departure from the Grover (2014) finding where prior math was a significant predictor of the post-score even when controlling for the pre-score. While one would have expected Grade 8 to perform better than Grade 7 (who would be expected to outperform Grade 6), our selections of schools and classrooms (teachers) made our results seem less surprising. Grades 6 and 8 were low performing schools, and as a result, started out lower and ended up lower than Grade 7 even though all three grades showed significant gains. The pace of the curriculum, as well as lack of strategies and curricular materials for differentiation across grades and abilities also meant that the pre-posttest as well as the curriculum may have been an unduly heavy lift for younger students and those with poor prior academic preparation in low-performing schools. This is discussed further in the Survey and Interviews analysis below.

Scratch projects

Qualitative analyses of 54 open-ended Scratch projects by VELA students and comparisons with the dataset (of *N* = 60 projects) from other district classrooms revealed that learners in our intervention clearly demonstrated better facility with the introductory programming concepts in their Scratch projects – their programs were more complex as compared to the comparison sample of projects collected from non-VELA study students and demonstrated better use of VELA constructs. Our salient findings from the comparison of Scratch projects are as follows.

As seen in Table 10, students' average total scores were significantly higher for VELA students' projects. Although average scores for "general factors", "mechanics of design", and "user experience" were not statistically different across the two groups, *VELA students' projects scored statistically significantly higher on "*basic coding constructs*" and "*advanced constructs*"*. VELA students'

Table 10. Two-sample t-tests of average scores on VELA and the non-VELA comparison final, open-ended projects of choice (gray rows indicate statistically significant differences).

	VELA Mean (SD) (*N* = 54)	Non-VELA Mean (SD) (*N* = 60)	*t*	p-value	Cohen's *d*
Total Score	96.0 (53.1)	71.8 (40.0)	−2.8	0.01	0.52
General Factors	31.4 (6.3)	29.2 (10.9)	−1.2	0.22	0.23
Mechanics of Design	17.5 (14.1)	16.4 (14.2)	−0.4	0.67	0.08
User Experience	4.6 (4.8)	3.4 (3.5)	−1.5	0.93	0.29
Basic Coding Constructs	31.83 (25.2)	18.17 (15.2)	−3.5	0.00	0.67
Advanced Constructs	10.72 (16.9)	4.6 (8)	−2.5	0.01	0.47
Variables	1.3 (2.9)	0.9 (1.0)	−1.6	0.10	0.31
Boolean Operators	2.8 (4.9)	0.9 (2.5)	−2.6	0.01	0.50
Relational Operators	4.3 (4.4)	2.7 (4.4)	−1.9	0.06	0.36
Repeat-Until Loops	1.7 ()	1.1 (2.1)	−1.3	0.21	0.24

average use of *Boolean operators in their free-choice projects was also statistically significantly higher than that of non-VELA students*. This is a significant finding, as our work consciously introduced this key concept that is often given little to no attention in middle school (Grover et al., 2018). "Variables", "Repeat-until" loops, and "Relational operators", also showed higher use on average in VELA projects than non-VELA projects, however, the difference was not statistically significant. Another interesting finding was the *spread of projects* that used "Variables", "Repeat-until" loops, "Relational operators" and "Boolean Operators". The number of VELA projects (as a percentage of *N* = 54) was significantly higher than the number of non-VELA projects (as a percentage of *N* = 60) for the use of *each of those foundational constructs and relevant focal VELA concepts* (Table 11). Lastly, grade-wise comparisons (Table 12) showed that, in general, grade 7 projects significantly outperformed the non-VELA projects. Also, grade 6 VELA projects showed significantly more use of "Basic coding constructs" and "Boolean Operators" than non-VELA grade 6 projects.

VELA students' results on this comparative evaluation of final Scratch projects are very encouraging, and especially their use of constructs related to our focal concepts. We realize that examining open-ended projects have their pros and cons. Open-ended projects do not always tell the whole story of student learning as students' free-choice projects may not showcase all that they have learned, or students may have received help and used code/constructs that they do not

Table 11. Percentage of VELA and non-VELA projects that used VELA constructs.

	VELA Projects that used these constructs	Non-VELA Projects that used these constructs	VELA % (out of 54)	Non-VELA % (out of 60)
Variables	40	29	74%	48%
Boolean Operators	22	14	41%	23%
Relational Operators	40	30	74%	50%
Repeat-Until Loops	20	17	37%	28%

Table 12. Grade-wise comparison of VELA and non-VELA projects.

	Grade 6 Mean (SD) *n* = 18	Grade 7 Mean (SD) *n* = 21	Grade 8 Mean (SD) *n* = 15	Grade 6 Mean (SD) *n* = 34	Grade 7 Mean (SD) *n* = 18	Grade 8 Mean (SD) *n* = 8
	VELA Sample (*N* = 54)			Non-VELA Sample (*N* = 60)		
Total Score	75.4 (35.0)	137.0*** (54.8)	63.4 (26.6)	70.9 (46.0)	75.9 (36.0)	66.8 (16.6)
General Factors	30.6 (6.8)	34.5* (6.1)	27.9 (3.4)	29.9 (12.8)	28.7 (9.1)	28.0 (3.8)
Mechanics of Design	10.9 (7.8)	28.9** (15.5)	9.5 (4.4)	17.8 (17.6)	15.4 (7.8)	12.2 (8.0)
User Experience	1.8 (2.2)	9.3*** (4.0)	1.4 (1.7)	3.0 (3.5)	4.3 (3.7)	3.3 (3.0)
Basic Coding Constructs	25.3* (15.3)	46.3** (30.9)	19.4 (14.8)	16.1 (16.2)	21.4 (14.9)	19.5 (10.6)
Advanced Constructs	6.8 (9.1)	18.0* (23.2)	5.2 (9.3)	4.0 (7.2)	6.2 (10.3)	3.8 (6.2)
Variables	1.3 (0.8)	1.3 (2.1)	1.1 (1.0)	1.0 (1.1)	0.61 (1.0)	1.0 (1.1)
Boolean Operators	2.6 (4.7)	4.7 (6.0)	0.2 (0.4)	.44 (1.7)	2.0 (3.8)	0.1 (0.4)
Relational Operators	3.4 (3.9)	6.5 (4.8)	2.3 (2.8)	2.0 (3.8)	4.4 (5.5)	2.0 (3.7)
Repeat-Until Loops	1.4* (2.1)	1.9 (3.6)	1.7 (2.9)	0.8 (1.8)	1.5 (2.8)	1.0 (1.8)

Note: Asterisks indicate significant difference between the score of VELA versus non-VELA group in the same grade (* p < .05; ** p < .01; *** p < .001)

understand well (Kurland & Pea, 1985). Finished projects, as Grover et al. (2017) explain, also do not show important aspects of learning that can be understood only from in-process evidence, for example how student debugged or approached the construction of a computational solution. On the flip side, a free-choice project showcases the complexity of construct use and programming structure that a student *chooses* to incorporate when no constraints have been imposed. Open-ended projects are considered important measures of learning and constitute an authentic form of assessment since children learn deeply when they create products with personal meaning that require understanding and application of knowledge (Barron & Darling-Hammond, 2008). Also, design activity involves stages of revisions as students define, create, assess, and redesign their products, and it often benefits from collaboration between students (Grover et al., 2015). Examining these artifacts thus bears the potential of providing valuable insight into the student learning experiences in introductory programming that can also point the way toward areas for improvement in introductory programming curricula (Grover et al., 2018). Additionally, our evaluation of the projects was very exhaustive. Even though construct frequency as a measure is not always adequate (Meerbaum-Salant et al., 2011), the rubric accommodates for duplication of code, and caps frequency count for most constructs in addition to accounting for several constructs only through presence or absence (1 or 0).

Lastly, grade-wise comparisons of the projects between the VELA and non-VELA sample groups provided insight into specific topics that VELA students learned better. Given our choice of schools and classrooms, grade 7 outperforming the non-VELA group was not entirely unexpected. However, it is significant that grade 6 VELA students had significantly higher scores for “basic coding constructs” as well as use of “Boolean operators” compared to the non-VELA grade 6 group given that grade 6 (and grade 8) VELA classrooms comprised several struggling learners compared to the average grade 6 or 8 classroom in the district.

Surveys and interviews

These aforementioned findings notwithstanding, the range of students' readiness to engage and learn, their instructional needs, and individual student reactions, varied considerably both within and across the three classrooms. The 6th-grade students had the most challenges, whereas the high-achieving 7th-grade class demonstrated most engagement with VELA ideas and curriculum.

This was evidenced in students' post-survey feedback on the digital activities (Table 13 and Figure 10). Overall, students found *Cats and Ladders* to be most engaging and fun. *Graphical Looping* was found to be too simple and

Table 13. Teacher and student reactions to some of the VELA *digital* activities.

Reactions to Cats and Ladders	(Teacher): "maybe it is having that tool that reaches the student on so many levels being that it is animated, there is sort of an emotional interest, all of that, so that story sticks out better. And so it may not really have huge payoff in that it shows you how to build an expression, but as the teacher emphasizes over and over again now that expressions are made from variables and operators and how that comes up later, it refers back to cats and ladders it is like oh yeah, cats and ladders, the two ladders, the two ladders. So the power becomes more on how the teacher refers back to the tool in later discussion in referencing the story plots".
	(7th grade student): "I thought it was interesting because this happens in real life, and I like cats, and I like the idea of saving them..I thought it was more challenging and interesting because you would have to not only type in the blue ladder height and orange ladder height, but also which building it was in. It made it more connected to realize because in the city that I live in, there's a lot of buildings".
	(8th grade student):"It's… to show us about height. And like the variables change. Because like you could change the numbers…It shows us variables. Like how high the ladder could go, like how we could change it."
	(6th grade student): "Cats & Ladders was fun and cool" **(a few other students had similar reactions)**
	(7th grade student): "I liked cats and ladders because it was like a real game".
Reactions to Graphical Looping	(8th grade student): "It's sort of like a puzzle…It teaches you the purpose of what loops do…It reminded like if we go to a grocery store and we could come back and get more food from there and then come back…Like each day like you like… it can be from like school, you go back home. And from the home you go back to school"
	(7th grade): "Graphical Looping was too easy and boring" **(a few other students had similar reactions)**
Reactions to Alarm Clock	(6th grade student): "I did not really understand it" **(a few students had similar reactions)**
	(8th grade student): "I liked alarm clock the most because it was most challenging"
	(8th grade student):"It was most interesting"
	(6th grade student): "It took too long"
	(Teacher): "And then the Boolean switch and the alarm clock one, I think are too big for the kids, as an activity to handle on their own. And so they become very teacher directed, and along that way you lose a lot of kids".
	(Teacher): "The interface is confusing, in particular with regard to drawing attention to which features are more important than others… Some kids might get it and understand what to ignore and what to work with, but it's not clear at first. And so even some of the naming schemes for what the boxes are, what the tables are, this table is how you're going to use this, and this table is how you're going to use that".
	(Teacher): "I mean I think it's a really cool project idea, but I think maybe just changing up the way kids enter in variables and expressions and stuff".

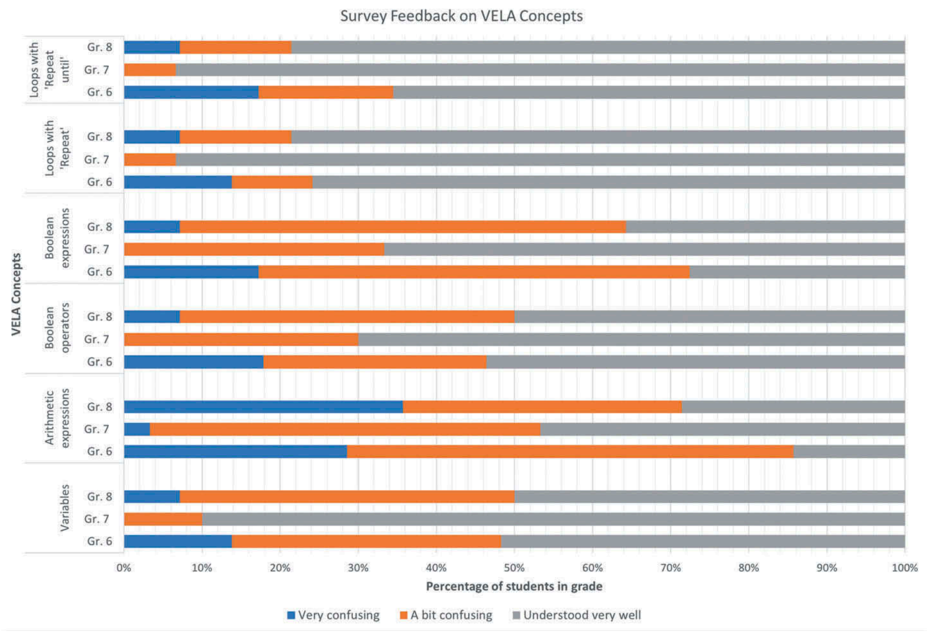


Figure 10. Survey feedback from students on their understanding of VELA concepts.

lacking in challenge. Some students found a few aspects of *Alarm Clock* to be confusing, although others indicated they learned from it.

Additionally, while students engaged well with variables, arithmetic expressions, and loops, there was relatively a greater struggle with abstraction and with Boolean variables and expressions (Figure 10). Some of these struggles appear to be related to experiences with the activities. Teachers noted in interviews, for example, that the expression-authoring features in the *Alarm Clock* and *Three Switches* microworld was complex and confusing for some students. However, one teacher also commented that although she found good bridging in Scratch lessons to VELA activities on variables, looping and conditionals, the connections to the expression abstraction work in *Alarm Clock* were not sufficiently revisited or reinforced in the subsequent Scratch programming activities. That said, one teacher commented on the deep value she saw in the VELA curriculum –

> Well for me, I felt like it really educated me quite a bit, deeply and powerfully. And so it was a huge learning process for me, not quite on the level of how to be a teacher in the classroom, but the content itself. So it developed my content knowledge really well. And I'm excited to now kind of apply it in the classroom, and try to marry our current curriculum to some of these ideas and flesh it out more. … I feel that this group learned programming deeper, by about 50% or more, if I could put a value on it, than previous groups I've taught, and so that their understanding of programming environments and Scratch is more robust than previous groups. And it's just an intuition of seeing it.(VELA Teacher)

Conclusion and next steps

Our current work in the context of middle school introductory programming draws on an approach in mathematics education that has succeeded in achieving better conceptual learning in middle school for diverse students. We created and examined dynamic digital interactives and microworlds that engage students with hard-to-learn programming concepts, finding particularly pedagogic value of dynamic representations to variability and abstraction in computing. We also drew from learning sciences research to scaffold novice learners' guided inquiry and exploration of programming concepts in a context separate from programming before they employ them in programs. Our empirical research suggests that these theories pertaining to conceptual learning (drawn from research outside of CS) can be put to work with success to aid the novice programmers' engagement with – and learning of – hard-to-learn concepts foundational to programming.

Our activity designs reflect several themes. For example, the *importance of naming* – of choosing names that are both meaningful to the problem context in which they are relevant, and productive in whatever logical or symbolic or verbal grammar will be used to reference and manipulate them – is one such essential theme, and recurs in our activities on *variables* and *expressions* (where names stand for changing quantities and for relationships between changing quantities), as well as on *abstraction* (where names stand as summaries for exhaustive definitions and detail). Another is that *story* is a productive and universally accessible metaphor and proxy for *program code* in a context in which learners are not yet able to grapple with code explicitly. Stories conveyed orally, through comic strips, or in writing – mirror typical code in having well-defined execution flows which are primarily sequential but occasionally undertake repetitive loops, and other control flows. Finally, dynamic representation design principles offer a ladder to abstract ideas and concepts emerging from concrete and manipulable representations that students find compelling.

Our results demonstrate the promise of introductory experiences with dynamic representations and relevant "story" examples and contexts for conceptual exploration. Overall, our students demonstrated learning gains in understanding of the VELA concepts as evidenced by pre-post assessment data analyses and an examination of final free-choice Scratch projects. More importantly, these learning gains were not significantly different by prior math preparation or gender or ethnicity. Our students' open-ended, free-choice projects scored significantly better on key dimensions (those more closely related to our focal foundational programming concepts) than projects from a comparison group.

This empirical research demonstrates the promise of our novel approach and curriculum for providing conceptual supports to novice programmers. Student and teacher feedback on the VELA activities as well as deeper item analyses of the assessment suggest areas for improvement in the next iteration of this DBR. Our next steps also involve improving the activity designs and building more robust curricular bridges between VELA and Scratch activities. Additionally, the role of the classroom teachers in the enactment of this curriculum (based on their own expertise and experience in teaching as well as conceptual understanding of CS) also plays an important role in the impact of a curriculum on student learning. Related ongoing research is examining individual student and teacher cases to better understand the learning process and the impact of diverse backgrounds and classroom environments. Future research will examine *how* specific elements of VELA activities contribute to student understandings for a more complete picture of the mechanics of learning in our novel approach to teaching introductory programming. We are also developing teacher professional development modules around the VELA curriculum to support deeper conceptual understanding of introductory programming concepts for teachers.

Acknowledgments

We thank the National Science Foundation for grant funding support (NSF #1543062). We are grateful to Jeremy Roschelle, Satabdi Basu, Matthias Hauswirth, and Bryan Twarek for their assistance and intellectual contributions, and to research staff at SRI International as well as teachers and students for their participation in this research.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work was supported by the National Science Foundation, USA, under Grant 1543062.

Notes on contributors

Shuchi Grover, Ph.D., a computer scientist and learning sciences researcher by training, is Chief Learning Scientist at Looking Glass Ventures in Palo Alto, California. Formerly senior research scientist at SRI International, Dr. Grover's current research focuses on computer science education especially introductory programming and the development of computational thinking skills in preK-12 learners.

Nicholas Jackiw is a senior research scientist at the Center for Education Research and Innovation, SRI International working in mathematics and computer science education. He also designs and develops software for learners of mathematics, where his work includes *The Geometer's Sketchpad* and *TouchCounts*.

Patrik Lundh, Ph.D., is a senior education researcher at the Center for Education Research and Innovation, SRI International. Dr. Lundh conducts research in K-12 science, mathematics, and computer science education in school and afterschool settings. He has extensive experience in qualitative research methods, design research, curriculum implementation and evaluation research.

ORCID

Shuchi Grover http://orcid.org/0000-0001-6633-8862

References

Ackermann, E. (2001). Piaget's constructivism, papert's constructionism: What's the difference. *Future of Learning Group Publication*, *5*(3), 438.

Barron, B., & Daring-Hammond, L. (2008). How can we teach for meaningful learning? In L. Daring-Hammond, B. Barron, P. D. Pearson, A. H. Schoenfeld, E. K. Stage, T. D. Zimmerman, … J. L. Tilson (Eds.), *Powerful learning: What we know about teaching for understanding* (pp. 1-10). San Francisco: Jossey-Bass.

Bransford, J. D., Brown, A., & Cocking, R. (2000). *How people learn: Mind, brain, experience, and school*. Washington, DC: National Research Council.

Bau, D., Gray, J., Kelleher, C., Sheldon, J., & Turbak, F. (2017). Learnable programming: blocks and beyond. *Communications of the ACM*, *60*(6), 72–80.

Du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, *2*(1), 57–73.

Ebrahimi, A. (1994). Novice programmer errors: Language constructs and plan composition. *International Journal of Human-Computer Studies*, *41*(4), 457–480.

Franklin, D., Skifstad, G., Rolock, R., Mehrotra, I., Ding, V., Hansen, A., … Harlow, D. (2017, March). Using upper-elementary student performance to understand conceptual sequencing in a blocks-based curriculum. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 231–236), Seattle, WA, USA. ACM.

Gobil, A. R. M., Shukor, Z., & Mohtar, I. A. (2009). Novice difficulties in selection structure. In International Conference on Electrical Engineering and Informatics, 2, 351–356. New Jersey, USA: IEEE.

Gray, D., Kelleher, J., Sheldon, C., & Turbak, F. (2017). Learnable programming: Blocks and beyond. *Communications of the Acm*, *60*(6), 72–80.

Grover, S. (2014). *Foundations for advancing computational thinking: balanced designs for deeper learning in an online computer science course for middle school students*. Stanford: Stanford University

Grover, S. (2019). An assessment for introductory programming concepts in middle school computer science. In *Proceedings of the 2019 Annual Meeting of the National Council on Measurement in Education (NCME)*, Toronto, CA.

Grover, S., & Basu, S. (2017). Measuring student learning in introductory block-based programming: examining misconceptions of loops, variables, and boolean logic. In Proceedings of the *48th ACM Technical Symposium on Computer Science Education (SIGCSE '17)*. Seattle, WA: ACM.

Grover, S., Basu, S., Bienkowski, M., Eagle, M., Diana, N., & Stamper, J. (2017). A framework for using hypothesis-driven approaches to support data-driven learning analytics in measuring computational thinking in block-based programming environments. *ACM Transactions on Computing Education (TOCE)*, *17*(3), 14.

Grover, S., Basu, S., & Schank, P. (2018). What we can learn about student learning from open-ended programming projects in middle school computer science. In *Proceedings of* *the 49th ACM Technical Symposium on Computer Science Education* (pp. 999–1004), Baltimore, MD, USA. ACM.

Grover, S., Jackiw, N., & Lundh, P. (n.d.). Non-programming digital interactives to advance learning of introductory computing concepts for diverse student populations. *Interactive Learning Environments*.

Grover, S., Lundh, P., & Jackiw, N. (2019). Non-programming activities for engagement with foundational concepts in introductory programming. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, Minneapolis, MN, USA. ACM.

Grover, S., & Pea, R. (2014). Expansive framing and preparation for future learning in middle-school computer science. In *International Conference of the Learning Sciences Conference,* Boulder, CO, USA. ISLS.

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, *25*(2), 199–237.

Grover, S., Pea, R., & Cooper, S. (2016). Factors influencing computer science learning in middle school. In *Proceedings of the 47th ACM Technical Symposium on Computer Science Education*. Memphis, TN: ACM.

Jackiw, N. (1991–2009). *The geometer's sketchpad (computer software V1, V5)*. Emeryville, CA: Key Curriculum Press.

Kurland, D., & Pea, R. (1985). Children's mental models of recursive LOGO programs. *Journal of Educational Computing Research*, *1*(2), 235–243.

Laborde, J. M., & Strässer, R. (1990). Cabri-géomètre: A microworld of geometry for guided discovery learning. *Zentralblatt für didaktik der athematic*, *90*(5), 171–177.

Lahtinen, E., Ala-Mutka, K., & Järvinen, H. M. (2005). A study of the difficulties of novice programmers. *ACM SIGCSE Bulletin*, *37*(3), 14–18.

Lewis, C. M., & Shah, N. (2012). Building upon and enriching grade four mathematics standards with programming curriculum. In *Proceedings of the 43rd Technical Symposium on CS Education*. Raleigh, NC: ACM.

Mayer, R. E. (2004). Should there be a three-strikes rule against pure discovery learning? *American Psychologist*, *59*(1), 14–19.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011, June). Habits of programming in scratch. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education* (pp. 168–172), Darmstadt, Germany. ACM.

Mislevy, R. J., Almond, R. G., & Lukas, J. F. (2003). A brief introduction to evidence-centered design. *ETS Research Report Series*, *2003*(1), i-29.

Papert, S. (1991). Situating constructionism. In I. Harel & S. Papert (Eds.), *Constructionism* (pp. 1-11). Ablex Publishing.

Pea, R., & Kurland, D. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology*, *2*, 137–168.

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., ., … Paterson, J. (2007). A survey of literature on the teaching of introductory programming. In *Proceedings of the 38th Technical Symposium on CS Education*, Covington, KY, USA. ACM.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, *13*(2), 137–172.

Roschelle, J., Shechtman, N., Tatar, D., Hegedus, S., Hopkins, B., Empson, S., … Gallagher, L. P. (2010). Integration of technology, curriculum, and professional development for advancing middle school mathematics: Three large-scale studies. *American Educational Research Journal*, *47*(4), 833–878.

Samurcay, R. (1989). The concept of variable in programming: Its meaning and use in problem-solving by novice programmers. In E. Soloway & J. C. Spohrer (Eds.), *Studying the novice programmer* (pp. 161–178). Psychology Press.

Schattschneider, D., & King, J. (1997). *Geometry turned on: Dynamic software in learning, teaching, and research*. Washington: MAA.

Schofield, E., Erlinger, M., & Dodds, Z. (2014). MyCS: CS for middle-years students and their teachers. In *Proceedings of the 45th ACM technical symposium on Computer science education*, Atlanta, GA, USA (pp. 337–342). ACM.

Schwartz, D. L., & Bransford, J. D. (1998). A time for telling. *Cognition and Instruction*, *16*(4), 475–5223.

Shute, V. J. (1991). Who is likely to acquire programming skills? *Journal of Educational Computing Research*, *7*(1), 1–24.

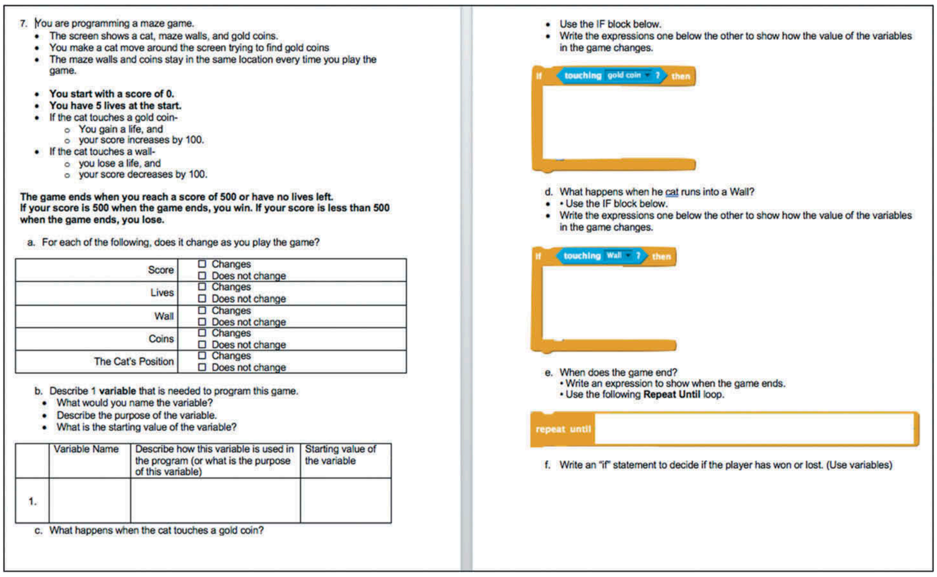
The White House. (2016). *Computer science for all*. Retrieved from http://bit.ly/2tcPrAj

Wang, F., & Hannafin, M. J. (2005). Design-based research and technology-enhanced learning environments. *Educational Technology Research and Development*, *53*(4), 5–23.

Wiggins, G. P., & McTighe, J. (2005). *Understanding by design*. Association for Supervision & Curriculum Development.

Appendix.

Sample item from design VELA pre-post assessment



7. You are programming a maze game.
   - The screen shows a cat, maze walls, and gold coins.
   - You make a cat move around the screen trying to find gold coins
   - The maze walls and coins stay in the same location every time you play the game.

   - **You start with a score of 0.**
   - **You have 5 lives at the start.**
   - If the cat touches a gold coin-
     - You gain a life, and
     - your score increases by 100.
   - If the cat touches a wall-
     - you lose a life, and
     - your score decreases by 100.

The game ends when you reach a score of 500 or have no lives left.
If your score is 500 when the game ends, you win. If your score is less than 500 when the game ends, you lose.

a. For each of the following, does it change as you play the game?

Score	☐ Changes ☐ Does not change
Lives	☐ Changes ☐ Does not change
Wall	☐ Changes ☐ Does not change
Coins	☐ Changes ☐ Does not change
The Cat's Position	☐ Changes ☐ Does not change

b. Describe 1 **variable** that is needed to program this game.
   - What would you name the variable?
   - Describe the purpose of the variable.
   - What is the starting value of the variable?

	Variable Name	Describe how this variable is used in the program (or what is the purpose of this variable)	Starting value of the variable
1.			

c. What happens when the cat touches a gold coin?
   - Use the IF block below.
   - Write the expressions one below the other to show how the value of the variables in the game changes.

d. What happens when he cat runs into a Wall?
   - • Use the IF block below.
   - Write the expressions one below the other to show how the value of the variables in the game changes.

e. When does the game end?
   • Write an expression to show when the game ends.
   • Use the following **Repeat Until** loop.

f. Write an "if" statement to decide if the player has won or lost. (Use variables)