Educational Technology Research and Development
https://doi.org/10.1007/s11423-018-9622-x

RESEARCH ARTICLE

What they learn when they learn coding: investigating cognitive domains and computer programming knowledge in young children

Amanda Strawhacker[1] · Marina Umaschi Bers[1]

Abstract
Computer programming for young children has grown in popularity among both educators and product developers, but still relatively little is known about what skills children are developing when they code. This study investigated $N = 57$ Kindergarten through second grade children's performance on a programming assessment after engaging in a 6-week curricular intervention. Children used the ScratchJr programming tool to create animated stories, collages, and games. At the end of the learning intervention, children were assessed on their knowledge of the ScratchJr language and underlying reasoning. Specifically, we explored children's errors on the assessment to determine evidence of domain-specific reasoning (e.g. mathematic, verbal, causal). Results show that while all students mastered foundational coding concepts, there were marked differences in performance and comprehension across the three grade levels. Interpretation of results suggests a developmental progression inherent in programming knowledge acquisition.; Implications for computer programming education and developmental research are discussed.

Keywords Early childhood education · Computer programming · ScratchJr · Cognitive developmental theory

Introduction

In recent years, the popularity of computer programming as a learning area has led to the development of many new tools, teaching approaches, and curricular interventions for young children (Kindergarten through second grade) to explore coding (Balanskat and Engelhardt 2015; K–12 Computer Science Framework 2016; Livingstone, 2012). Platforms like Tynker (https://www.tynker.com/), Hopscotch (http://www.gethopscotch.com/), and Move the Turtle (http://movetheturtle.com/) promise to engage children in computer

✉ Amanda Strawhacker
Amanda.Strawhacker@tufts.edu

Marina Umaschi Bers
Marina.Bers@tufts.edu

[1] DevTech Research Group, Eliot-Pearson Department of Child Study and Human Development, Tufts University, 105 College Ave, Medford, MA 02155, USA

Published online: 20 August 2018

coding at a level fitting their cognitive needs, and organizations like Code.org (http://code.org/) and Code Academy (http://www.codecademy.com/) integrate foundational coding principles into interactive lessons for all ages, starting with Kindergarten. Prior research in early childhood settings has shown the benefits of introducing technology and engineering early for improving children's sequencing ability, design learning, executive functioning, mathematic and linguistic development, and logical reasoning (Bers 2008; Clements and Sarama 2003; Flannery and Bers 2013; Kazakoff 2014; Kazakoff and Bers 2012; Kazakoff et al. 2013; Wyeth 2008). However, relatively little research has been conducted on programming as a learning area in itself, and which existing knowledge domains are cultivated when young children engage in programming.

This study describes results from an assessment of $N = 57$ K-2nd grade students' understanding of foundational programming concepts after completing an introductory programming learning intervention. Average scores are reported, and common errors are explored for evidence domain-specific logic from areas such as verbal, spatial, and quantitative domains. Implications for future research and for programming learning in early childhood are discussed.

Cognitive development in young children

Piaget (1953) famously argued that children develop intelligence by naturally progressing through a series of large bio-psycho shifts that alter a wide range of the child's mental models (e.g. when a child understands that when she cannot see someone, they continue to exist). He theorized that these large shifts are the basis for several stages of intelligent development. Children in the current study ranged from 5 to 8 years of age. At this age, children are moving from the concrete operations stage to the pre-operational stage, meaning that at the younger end, they rely on physical symbols and representation, and as they get older, they are better able to take others' perspectives, use logic-based causal reasoning, and rely less directly on physical representations of ideas (Feldman 2004; Lightfoot et al. 2009; Piaget 1953; McDevitt and Ormrod 2002).

Most developmental researchers agree that as children learn and grow, unique domains of knowledge emerge in their thinking (Case 1992; Demetriou 2000; Fischer 1980; Feldman 1988; Piaget 1953). In their seminal discussion of the architecture of a child's cognitive system, Demetriou et al. (2010) discuss domains of knowledge that have emerged in their extensive clinical-trial research on children's cognitive capacity. These domains include categorical, quantitative, spatial, causal, social, and verbal knowledge. The authors hope to reveal insights about which of these established learning domains young children exercise when coding by examining their responses to a standardized computer programming assessment. In any assessment of young children's knowledge, the type and frequency of mistakes can provide clues about the nature of the participant's content understanding (or lack thereof). For example, Piaget's classic conservation task can reveal a child's level of concrete awareness, and Karmiloff-Smith's block-balancing task provides insight into children's theories-in-action as they attempt to master the task (Elkind 1961; Karmiloff-Smith and Inhelder 1975). The authors hypothesize that children's preliminary theories about programming may also be revealed by exploring their responses on a programming task. In the current study, children's responses to a standardized programming assessment are used to identify the domains of knowledge that contribute to children's programming strategies. In the next section, we describe empirically-established domains that informed our analysis.

Existing knowledge domains

In an attempt to merge accepted principles from developmental psychology and neo-Piagetian theory, Demetriou and Case theorized a three-level organization of mental processing, with one level to represent the underlying mental ability to process thoughts within the mind, a second level to process information coming from the outside environment, and a third level to monitor and regulate thought processing (Case et al. 2001; Demetriou et al. 2010). Specifically, within the second level, which pertains to learning and knowledge processes involving representational objects, Demetriou and colleagues proposed six domains of knowledge that young children demonstrate: categorical, quantitative, spatial, causal, social, and verbal (Demetriou et al. 2010). We will use these domains to contextualize findings, and for the duration of the paper will refer to them interchangeably as "learning domains" or "knowledge domains."

Demetriou's six domains each have their own specialized functions, with a meaningful symbol system and unique objects and processes that are particular to that domain (e.g. spelling a word with letters does not rely on quantitative operations). Categorical thinking involves identifying and working with similarities and differences, such as organizing objects by color, and is foundational to inductive inference. Quantitative thinking is any mental operation involving quantitative transformations. This involves things like specifying the number of items in a small set without list-counting (e.g. 3 or 4 items), counting large numbers, and arithmetic operations of adding, subtracting, multiplying, and dividing. Spatial thinking involves mental representations of objects in space, both within objects (e.g. perceptions of structure, composition, size, depth) and between objects (e.g. distances, locations, directions, rotations). Causal reasoning has to do with logical dynamics, and involves if–then sequences, transfers of energy, and trial and error tactics to assess those relationships. Social thinking deals with understanding social relations, cultural norms and rules, facial expressions, and processes of imitation and perspective taking. Finally, verbal reasoning facilitates interactions between people, describing intentions or plans, and using grammatical and syntactical structure to organize processes and relationships in the other knowledge domains (e.g. the verbal phrase "if…then…" to describe causal relationships). Demetriou argued that these domains work synchronously to inform a person's general intelligence, but the differentiation of domains contributes to the organization of the growing mind.

In this study, we used these domains to characterize what we call "programming knowledge", or the thinking processes that young children demonstrate as they first explore computer programming. These domains, each with their own unique symbol systems and mental manipulations, likely afford different strategies to interpret the symbol system of a novel programming language. For example, a child attempting to use a verbal approach might apply narrative elements, names, or definitions to programming symbols, while a child using a mathematical strategy might make predictions about what will happen if they change the number parameter on a character's Move Forward block. Children who leverage social domains might ascribe internal motivations to their animated characters prior to coding them, or they might work collaboratively with a peer to create matching projects. More research is needed to understand how young children can meaningfully engage with foundational symbols and concepts specific to programming. The current study aims to address this gap in the empirical research literature, by investigating children's responses to programming logic questions, and seeking evidence of strategies from any of the pre-existing domains identified by Demetrious, et al. (2010).

The role of computer programming in cognitive development

We know that children rely on their senses to understand the world around them, and can be confused by processes that are invisible or intangible (Piaget 1953; Karmiloff-Smith and Inhelder 1975). Today, however, children are not just exposed to physical tools and objects, but also to digital and virtual ones. Papert (1980) developed his theory of Constructionism to describe how children construct knowledge using technology. In digital worlds children are not only manipulating objects, they are creating them; not just learning and testing rules, but writing them. By constructing, revising, and sharing artifacts in virtual environments, children are simultaneously constructing and revising their own knowledge (1980). However, researchers still do not know what kind of knowledge children construct when engaged in programming. In other words, is programming a wholly new kind of skill that never existed before the advent of programming languages? Or, is it a mix of pre-existing knowledge domains and skills that children have learned for many generations, adapted to apply to programming?

Prior research has focused on educational computer programming as a medium to develop foundational skills of math, logic, and sequential ordering (Kazakoff et al. 2013; Wyeth 2008; Kazakoff and Bers 2014; Pea and Kurland 1984; Clements 2002). Programming is traditionally associated with these kinds of skills, since it can require logic-based, detail-oriented thinking (Robins et al. 2003). However, there is evidence that a focus on programming as mainly quantitative or logical is too narrow (Fletcher-Flinn and Gravatt 1995). Resnick (2006) reminds us of the unique power of computers to virtually represent creative projects that would be difficult or impossible to realize in the physical world. A comprehensive review of literature reveals that computer-based learning experiences can promote children's ability to reflect on their own mathematical intuitions and ideas, catalyze science discovery by allowing students to engage in digital simulations, foster gains in verbal skills and creativity in 4- and 5-year-old children, and increase social and linguistic communication among young children (Clements and Sarama 2003; Fletcher-Flinn and Gravatt 1995). This suggests that programming is either a global thinking process that supports developmental skills foundational to other domains, or perhaps that those other skills (social, linguistic, scientific) are all deployed when a child engages in a programming task. This question is at the core of the current study, and little research addresses this point in the context of programming.

Bers (2018) writes that the public discourse relies too heavily upon a "problem solving" metaphor when designing coding tools for children. She writes that most programming tools marketed for young children emphasize coding as a logic game, and are built like simple puzzles to solve. Bers suggests a metaphor of expression is more appropriate for fostering creative learning (2018). ScratchJr is an introductory programming language that enables young children (ages 5–7 years) to explore fundamental computer programming concepts through creative expression. ScratchJr is unique among programming environments because it is designed to allow children to create the own digital objects, build animations, and tell stories through coding. Although many apps may reveal children's ability to work out logic puzzles, ScratchJr provides an opportunity to explore how children build and interpret complex systems of code with meaningful story lines and characters. This kind of coding provides a view into children's programming understanding that is richer than what can be ascertained from puzzles. For this reason, ScratchJr is the medium in this study to explore children's programming learning.

Technology as developmental learning tool: empirical research

A small number of research studies have examined programming knowledge in the context of theorized developmental stages. In a meta-analysis of early empirical work on adult programmers, Davies (1993) argued that programming knowledge can be described by "stereotypic knowledge structures," and which mental representations and knowledge structures are used is largely determined by the programming language taught. However, he also acknowledged that certain skills appeared to be generalizable across many programming languages (i.e. programmers who are experienced in many languages exhibit similar skill sets), and concluded that perhaps the method of instruction also plays a role in a programmer's mental representations (Davies 1993). Mioduser et al. (2009) investigated Kindergarten children's utterances and mental representations when observing a robotic program being executed. Initially, children's explanations were animistic descriptions of what the robot "wanted" to do. As children developed more sophisticated understandings of the robot's program, their explanations became increasingly generalized and more focused on the patterns and rules in the robot's behavior, suggesting that they began to leverage more concrete-abstraction in their mental representation.

More recently, researchers have attempted to understand children's developmental readiness for programming, in order to better inform the design of languages and curricular interventions. Flannery and Bers (2013) conducted an investigation of children ages 5–7 in which they categorized a child's developmental stage (which roughly correlated with age), and then observed them solving a robotic programming task (program a robot to dance the Hokey-Pokey). They identified the following two key indicators of programming knowledge in children: (1) ability to match a programming command with its outcome or action, and (2) ability to construct a program that uses the correct commands in the correct order. They found that the earlier a child's developmental stage, the less likely they were to understand a programming command's outcome, or to complete the open-ended programming task. In a similar study, Vizner (2017) investigated children's robotic programming practices and organized the behavior he observed into a developmentally-informed hierarchy, ranging from pre-programmer to fluent programmer. This hierarchy revealed two findings: first, that children in one developmental stage may demonstrate many different levels of programming behaviors (in other words, a child's developmental stage does not necessarily predict how quickly they will progress through the programming behaviors hierarchy); and second, that children do not often exhibit programming behaviors in a linear trajectory, and instead will often exhibit behaviors from different levels of the hierarchy during the same programming task (Vizner 2017).

Although these studies shed some light on the abilities of children and adults to acquire programming skills, there is still a lack of empirical work that investigates what constitutes programming knowledge. Davies (1993) and Mioduser et al. (2009) focused on how programming techniques and mental representations emerge, while Flannery and Bers (2013) and Vizner (2017) studied children's developmental progression through programming without linking to other developing knowledge domains. This study seeks to unite the two approaches. We first administered an assessment to capture the two programming knowledge constructs identified by Flannery and Bers (2013), symbol recognition and sequencing. Then, like Mioduser et al. (2009), we examined children's finished programs and common errors for evidence of their mental representations, and applied Davies' (1993) question about which (if any) pre-existing knowledge domains are suggested in their responses.

Purpose of the current study

This study sheds light on the cognitive domains that young children leverage when learning programming for the first time. Researchers examined Kindergarten children's responses to a standardized programming assessment after a learning intervention, in order to identify patterns in their responses that would suggest one or more pre-existing knowledge domain (e.g. verbal, mathematical, causal). The authors hypothesize that children's responses will suggest a diverse range of domains beyond quantitative and causal, which are often associated with learning programming, suggesting that current practices for teaching programming may be too narrow in scope.

In this paper, we will first address the following research questions:

1. What evidence of children's programming strategies can we infer by analyzing children's responses and common errors on a standardized programming assessment?
2. What strategies from existing knowledge domains (e.g. language, visual-spatial, etc.) are evident as contributors to children's programming logic?

These questions will be addressed by describing trends in student's answers to a standardized programming assessment. Following this analysis, we will interpret findings based on prior research into pre-existing knowledge domains (as identified by Demetriou et al. 2010).

Since this investigation is rooted in a stagewise theory of cognitive development, the authors hypothesize that some differences will arise across grades. The authors further expect to find evidence that children engage in a relatively uniform developmental progression when learning to program, separate from but informed by their development stage in other knowledge domains.

Method

In this mixed quantitative and qualitative case study, children in Kindergarten through 2nd grade engaged in a six-week learning intervention using the ScratchJr programming environment. A suburban public school in the greater New England area of the US was chosen as the research site, primarily because the school uses a "one iPad per child" policy, which allows every student to have access to an iPad throughout the day during relevant class sessions. One Kindergarten, one first grade, and one second grade classroom were purposively selected based on the relevance of this age range to the pedagogy under study. The classes comprised a convenience sample of (N = 57) K-2nd grade participant children (n = 27 male, n = 30 female). Data was collected in the form of children's responses on a post-intervention assessment of programming comprehension and knowledge. Responses were numerically scored against a set of correct answers, and trends in scores are reported. Incorrect answers were examined for qualitative trends in responses.

Learning intervention and pedagogical approach

Children in this study participated in a constructionist learning environment that fostered open exploration and self-directed project creation (Papert 1980; Bers 2014). This pedagogical model was chosen because of its success in other pilot studies on children's computer programming learning (Flannery et al. 2013; Kafai and Resnick 1996). Children were able to experiment with the foundational programming concepts needed to build an

interactive project. The intervention was conducted in students' regular classrooms, with researchers leading lessons and classroom teachers assisting and collaborating throughout. ScratchJr was introduced in twice-weekly 1-h lessons over 6 weeks, for a total of 12 classroom hours. During ScratchJr classes, students were encouraged to move around the room, collaborating and sharing knowledge informally (see Fig. 1). Each student worked one-on-one with his or her school-issued iPad tablet, and ScratchJr classwork was saved locally to that device.

Researchers worked closely with lead classroom teachers to implement the lesson activities. Kindergarteners did not complete all of the lessons in the most advanced module, and were not tested on certain concepts in the programming assessments.

ScratchJr: a programming environment for young children

The ScratchJr software was chosen as the intervention tool for this study due to its open-endedness as a learning platform for children to explore computer programming. A discussion of ScratchJr is warranted here, as interpretation of the assessment results depends on a basic understanding of the programming tool.

The ScratchJr programming environment is the product of an NSF-funded (DRL 1118664) research collaboration between the Lifelong Kindergarten Group at MIT, the Developmental Technologies Research Group at Tufts University, and the Playful Invention Company (http://www.scratchjr.org). ScratchJr is a tablet-based programming environment inspired by the MIT Lifelong Kindergarten group's popular Scratch programming language (http://scratch.mit.edu) for children ages eight and older, but with interface and programming language adjustments to make it developmentally appropriate for children in Kindergarten through second grade (Flannery et al. 2013). The layout of a ScratchJr project consists of four parts: a programming "editor," or workspace to drag and connect programming; a stage where characters act out the instructions on the programming blocks; a list of all the characters on the stage; and a gallery of up to four pages, each with its own stage, workspace, and character list to continue the project (see Fig. 2).

Children use the programming blocks to create programs that make characters move, hop, dance, and sing. These programs can range from very simple instructions (e.g. a single character growing in size) to quite complex (e.g. two or more characters interacting and conversing in dynamic settings). Because there are hundreds of potential combinations of the 28 programming blocks, ScratchJr projects are usually quite unique. This makes it a well-suited platform to focus on diverse learning domains, since ScratchJr is an inviting, child-friendly programming environment with a power reminiscent of more advanced languages.

Lesson content

The lesson activities used throughout this study are adapted from the Animated Genres curriculum, designed by the Developmental Technologies Research Group (Portelance et al. 2015) and freely available on the ScratchJr website (www.scratchjr.org). This curriculum contains several lessons which introduce foundational ideas from engineering desogn and computer science. Children engaged with lesson themes through structured classroom lessons, semi-structured games and activities, and open-ended programming play.

Fig. 1 During ScratchJr lessons, children moved freely around the room, collaborating informally

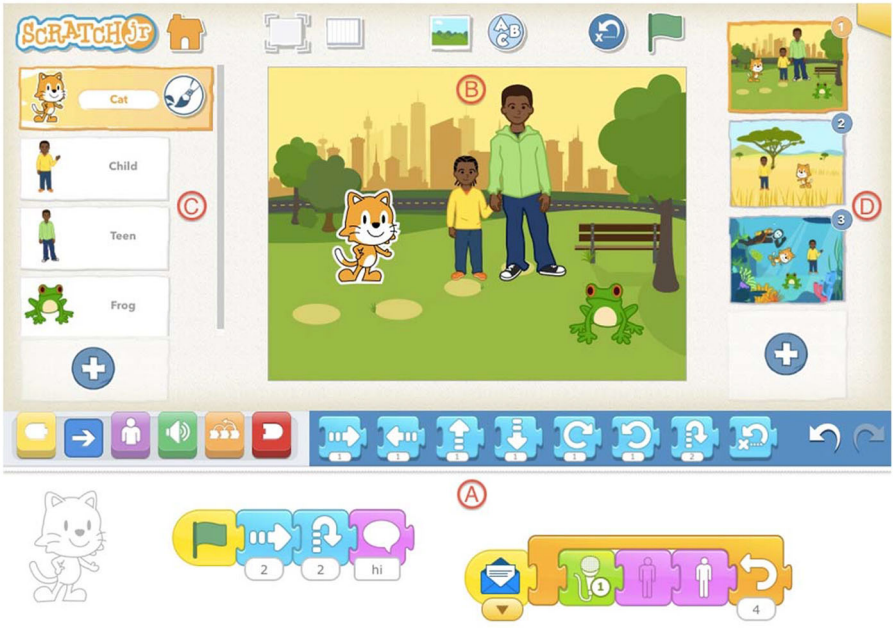


Fig. 2 A screenshot of the ScratchJr interface. The interface consists of **a** a programming "editor," or workspace, **b** a stage where characters execute programs, **c** a list of all the characters on the stage, and **d** a gallery of up to four pages, each with a new stage, workspace, and character list to continue the project

The lessons comprised three modules based on three genres of programmed media: interactive collage, animated story, and interactive game. Each module included a series of lessons on the unique characteristics of each genre, and an introduction to relevant ScratchJr features and programming blocks (see Table 1). Children concluded each module by creating an expressive work in that genre.

Measurement

Children's programming knowledge was assessed using experimental pilot tasks designed to capture programming comprehension and fluency with the ScratchJr icon-based language. As Davies (1993) pointed out, the programming language that programmers learn with can inform the mental representations that children use while programming. For this reason, it was imperative to use a ScratchJr-specific programming assessment to understand children's logic while answering questions. Readers may wonder why the learning method involved open-ended exploration, but the tasks seemed like straightforward problem-solving questions. Although each question does present a problem to be solved, these tasks allow for a diversity of creative, open-ended solutions. The open-endedness in

Table 1 Animated genres curriculum

Module[a]	Genre skills	ScratchJr skills
(1) Interactive collage	Set a scene by choosing backgrounds, characters, and actions Choose actions that would surprise or delight a novel viewer of the collage (i.e. user-centered design)	Navigate the ScratchJr interface Name projects Create ScratchJr backgrounds and characters Use ScratchJr blocks to program what characters do, including movement and appearance Read a program, and explain what it says, which character will move, and how the program will start (e.g. press a button, tap the character, etc.)
(2) Animated story	Use characters and backgrounds to tell a story Consider the sequence of actions, dialogue, or new backgrounds that best conveys an idea Identify the beginning, middle, and end of familiar and new stories	Use ScratchJr blocks to program transitions in the story, such as page turning Use ScratchJr blocks to program how characters do their actions, including setting speeds and using repeat loops Record sounds and use on-screen text to add narrative elements and character expression Consider the sequence of blocks and the impact on the sequence of events in the story Coordinate multiple characters to program conversations and reactions
(3) Interactive Game	Plan and incorporate clear instructions for playing a game Identify multiple ways to play a game that would allow some players to win and some players to lose Build choices and varied consequences into a game	Program transitions or reactions based on a triggering event (e.g. characters should react when touching another character, or when tapped by a user) Program different transitions or reactions based on different triggering events, to create "win" and "lose" events

Summary of each module and core content skills covered in the animated genres curriculum

[a]Each module consists of four 1-h lessons over 2 weeks

responses was necessary so that children could perform logical steps to complete the tasks, giving researchers a wide array of responses to analyze. These tasks were directly inspired by previous work on robotic programming assessments called Solve It tasks, which required children ages 5-7 years to construct programs to correspond with a story about a robot (Strawhacker and Bers 2015; Sullivan and Bers 2013). The experimental tasks developed in this study are very similar to the pilot assessment that inspired it, with several key differences described below.

Solve its: assessing children's programming knowledge

The ScratchJr Solve Its used in this study are open-ended tasks that rely mainly on reverse-engineering a program that corresponds to a finished project that children view on a projected screen. The tasks themselves are more complicated that straightforward puzzle-solving, as they require children to observe, remember, and reason about what they saw in the project. In this study, researchers were most interested in (1) whether children understood the prompts, and (2) what kind of errors children made while completing the Solve Its, as errors suggest the kinds of logical connections they were trying to make while completing the task.

Recall that Flannery and Bers (2013) identified two indicators of programming knowledge in children: (1) ability to match a programming command with its outcome or action, and (2) ability to construct a program that uses the correct commands in the correct order. The Solve It tasks assess both of these indicators, and take Flannery and Bers' work further, by analyzing mistakes as well as proportion of correctly-used commands. Solve Its are divided into two types: Block Recognition tasks (matching a command with its outcome) and Block Sequencing tasks (constructing a program with correct commands in the correct order). Although they employ different task types (roughly equivalent to multiple choice and open-response question types), they both use the same animated prompt (for more information on Solve It tasks, view prompts at: https://www.youtube.com/embed/l9d9L9hc2o4?list=PLsXbKpJZTa6P2bt7GpZTAZuvZT1cmFKa5. See also Appendix A: Answer Key for Solve It tasks, and Appendix B: Scoring Rubric for Solve It tasks).

In Block Recognition tasks, children were shown a short animation of one or more ScratchJr characters, but the corresponding commands used to construct that animation remain hidden. Then, using a paper print-out with all of the ScratchJr programming commands listed, they are asked to circle the blocks they believe were used to create the program they observed (see Fig. 3). For example, if a child is shown an animation of a cat hopping, she might recognize that the cat's program included a Hop block, and circle that on her assessment sheet. A total of six Block Recognition Solve Its was administered to each child in 1st and 2nd grade, and five Solve Its to Kindergarten students (see Table 2). Children completed all Block Recognition tasks before moving on to Block Sequencing tasks.

In Block Sequencing tasks, children viewed a ScratchJr project and then, using paper slips with ScratchJr blocks on them, constructed a program to match the project they just viewed (see Fig. 4). Due to time constraints, and limits on children's ability to maintain focused attention on the task, a total of three Block Sequencing Solve Its was administered to each child in Kindergarten, first grade, and second grade (see Table 2). Children viewed three of the same projects that they had seen earlier in Block Recognition tasks 4, 5, and 6 (see Table 2). These tasks were chosen because they assessed ScratchJr topics that ranged progressively from easier to more difficult (Flannery et al. 2013), but also because they

Fig. 3 A completed block recognition assessment sheet. Circled blocks represent the student's answers for viewing Solve It #4, #5, etc

allowed for progressively more flexibility and open-endedness in responses. For example, some questions allowed children to select two correctly sequenced programs from four choices, letting children ignore the task of command selection, while other questions required choosing and sequencing blocks. This means that certain steps in children's programming logic were held constant at different times, allowing researchers to more clearly understand gaps in children's knowledge.

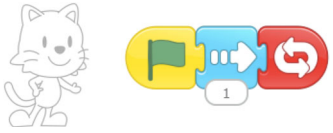
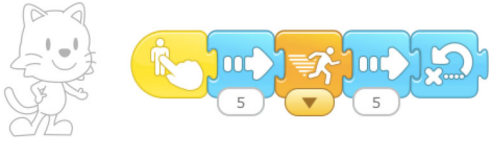
There are two key differences between Strawhacker and Bers' (2015) robotic pilot assessment and the one used in the current study. First, children in the earlier study listened to a researcher read aloud the story prompt, but children in the ScratchJr study viewed a project on a screen. It is possible that using a visual versus a verbal/auditory prompt had a significant impact on children's ability to understand the task. If the visual nature of the prompt shaped a child's mental representation of the program, one might expect to find specific types of errors in her responses. For instance, she might confuse programming blocks that look like they might create the visual result, but in fact function differently, such as the Move Right and Change Speed blocks (see Fig. 5). Indeed, such errors were observed in this study, although potential alternative causes are considered in the discussion section.

Another key difference is that children in the robotic pilot study were only given the programming blocks that they would need to complete the task. In this study, all of the block icons in the ScratchJr language were available for all tasks. Although this technique makes it more difficult to isolate sequencing ability, it allows researchers to compare children's responses in both Block Recognition and Block Sequencing on the same Solve It task.

Table 2 Description of Solve It tasks

Solve It task	Block recognition assessed	Block sequencing assessed	Grades assessed	Difficulty level[a]	Concept assessed	Correct programs
Task #1	X		K, 1, 2	Easy	Motion blocks	
Task #2	X		1, 2	Hard	Control flow: message passing interactions	
					Control flow: repeat loops	
Task #3	X		K, 1, 2	Medium	Control flow: timing	
Task #4	X	X	K, 1, 2	Easy	Looks blocks: resizing	

Table 2 continued

Solve It task	Block recognition assessed	Block sequencing assessed	Grades assessed	Difficulty level[a]	Concept assessed	Correct programs
Task #5	X	X	K, 1, 2	Hard	Control flow: endless loops	
					Triggering commands: start on bump	
Task #6	X	X	K, 1, 2	Medium	Control flow: speed	

[a]"Difficulty level" was arrived at by consensus of the research team and informed by pilot ScratchJr curriculum research (Flannery et al. 2013)

Fig. 4 A completed sequencing assessment task. The child chose these icons from an assortment of all the blocks available in ScratchJr, and ordered them into a sequence. This sequence represents the child's guess about the program that would have caused the project they viewed in Solve It Task #4

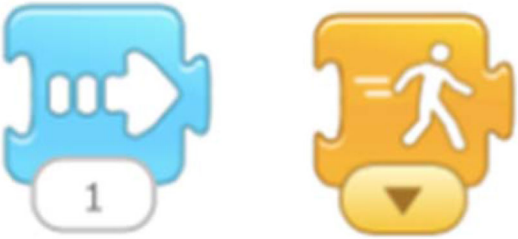


Fig. 5 ScratchJr's Move Forward block (left) and the Speed block (right) both look as if they might be commands to make a character run to the right

Table 2 shows all of the programs that children viewed in order to complete their Solve It tasks. Questions were first created, and then organized by the research team into Easy, Medium, and Hard questions based on the complexity of the programming task and the cognitive capacity required to complete them. These determinations were based on developmental stage theory (Piaget 1953). In order to heed on Davies' (1993) caution that mental representations in programming are directly informed by the language of instruction, prior findings from ScratchJr pilot studies with K-2nd grade students were also used to develop the questions (Flannery et al. 2013). Strong patterns emerged throughout the initial pilot research of ScratchJr, consistent with our assessment of easy vs difficult tasks from a cognitive developmental perspective. For example, children more easily mastered motion, sound, and look blocks, which all correspond with visual changes in the animation and thus relate to younger children's concrete operations (Piaget 1953). Conversely, they had more difficulty when programming multiple characters, multiple scripts within characters, or with control-flow blocks, which rely more on later-childhood abstraction (Piaget 1953; Mioduser et al. 2009).These trends helped shape the Solve It assessments, and questions were designed to target many of the observed differences in programming ability from high- to low-mastery students (see Table 2).

Scoring Solve It assessments

In the Solve It assessment, a perfect score is a score of zero. Raw number scores were given based on the number of steps required to change a child's response to the nearest correct program. For example, a child who missed a needed block (1 point) and circled an incorrect block (1 point) would receive a score of 2 points. Several patterns emerged in the data involving commonly confused blocks. Some blocks were mistaken for others with

similar programmatic functions, such as when a child circled both a Move Up and a Move Down block when they observed a character using the Hop block (to view the Solve It task video prompts, visit: https://www.youtube.com/embed/l9d9L9hc2o4?list=PLsXbKpJZTa6P2bt7GpZTAZuvZT1cmFKa5. See Appendix A: Answer Key for Solve It tasks, and Appendix B: Scoring Rubric for Solve It tasks for more information regarding the scoring process). Other confusions grew from the block icons, for example when a child chose a Spin Left instead of a Spin Right block. Some specific errors were common enough (i.e. present in more than half of the sample) that the research team altered the scoring procedure to minimize error inflation.

Since several Solve It tasks could be recreated with multiple correct programs, researchers scored student responses against the program that was closest to the child's answer. For example, one Solve It task showed a cat moving to the right multiple times. Some children circled the Move Right block, while others circled Move Right and Repeat Loop (see Table 3). Some limitations were introduced, however. For example, in the same Solve It task, a student could technically make a correct program using a Move Left block and negative number parameters, causing the cat to walk backwards. Since students had not covered negative numbers in either their ScratchJr or regular math lessons (and the cat would be facing the wrong direction anyway), this was considered an incorrect answer, and graded against the nearest correct answer (see Table 3).

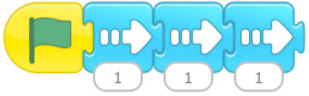
Reliability and validity

Assessments were scored using a rubric developed on a subset of the children's responses. Consensus coding was employed during data analysis of the Block Recognition data, with one researcher conducting an initial pass of coding and a final version being determined through conversations with the research team (Corbin and Strauss 2008). Although the Solve It data is quantitative, qualitative interpretations were necessary to understand children's logic errors. All researchers involved in scoring responses spent significant time using the ScratchJr environment, in order ensure that researchers had a common understanding of possible errors. Cohen's kappa (κ) was run to determine the level of agreement between two researchers trained in scoring the Block Recognition assessment. A test dataset of 42 sample Block Recognition responses was generated by the lead researcher, based on a 15% subset of students responses in the sample. This dataset was then scored by two research assistants. There was significant agreement between the two raters' judgements, $\kappa = .78$, $p < .001$. Researchers found difficulty reaching scoring consensus about the Block Sequencing data. For this reason, descriptive Block Sequencing results will be presented here as a way triangulate results from Block Recognition responses. All questions and all responses are expressed within the language and visual platform of ScratchJr, indicating construct validity of the assessment. Results from the analysis of errors also revealed certain predictable patterns that emerged in responses, lending internal validity to the assessment as well.

Results

In this section, we address our first research question, "What evidence of children's programming strategies can we infer by analyzing children's responses and common errors on a standardized programming assessment?" Solve It assessment scores were compiled and analyzed for trends. Following this, we explore the second question, "What strategies from existing knowledge domains (e.g. language, visual-spatial, etc.) are evident as

Table 3 Sample programs made with circled-block responses to "Solve It Task 1: Moving a Character Forward."

Correct programs using two blocks	Correct programs using three blocks	Incorrect programs
		
		

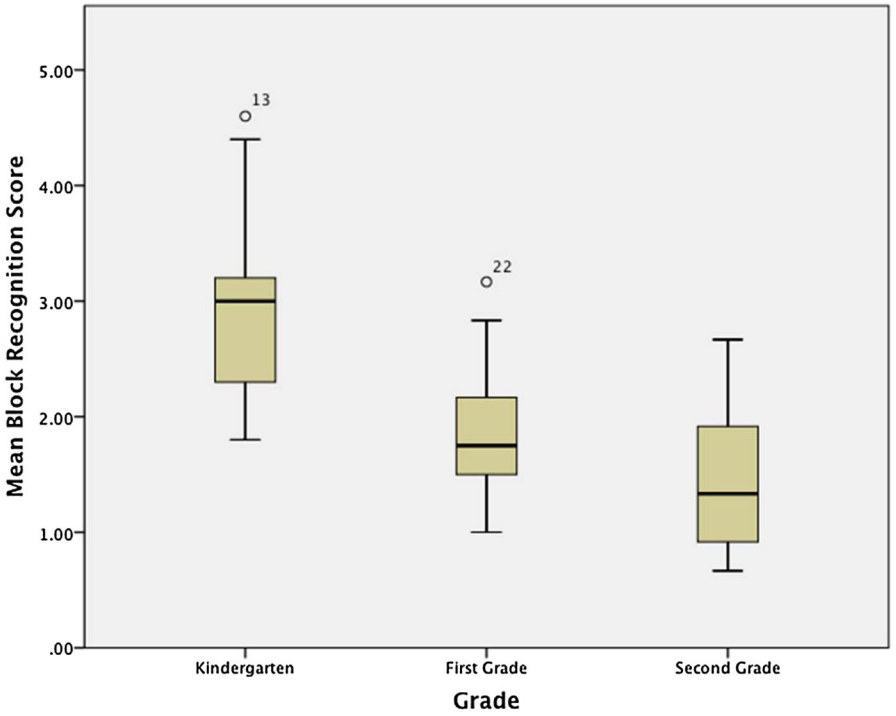


Fig. 6 Boxplots showing distributions by grade of mean scores on block recognition assessment

contributors to children's programming logic?" Results will be followed by interpretation in the discussion section.

Preliminary analysis

Taken together, the data showed no violation of normality, linearity, or homoscedasticity. After testing for outliers using boxplots and skewness values, three extreme outliers emerged from the data. Based on researchers' field notes, these outliers probably reflect a combination of several factors, including children's attention limits. These outliers were removed, bringing the sample size to $N = 55$ students ($n = 29$ females, $n = 26$ males).

Looking at grade-wise trends shrunk the sample sizes of each statistical test. To account for the small sample size, non-parametric tests were used when grades were segmented for analysis. As mentioned above, some extreme outliers were eliminated from the data, which revealed two emergent outliers (see Fig. 6). Because the Kruskal–Wallis test is robust to outliers, the results including the two emergent outlying data points will be discussed here. Histograms of the score distributions in each grade were relatively normally distributed, with skewness values falling between 1 and -1 for all groups. Finally, Levene's test for equality of variances was found to be non-significant, $F(2, 51) = 0.203$, $p > .05$ justifying the use of Kruskal–Wallis to evaluate the data.

Looking at individual Solve It tasks also resulted in a new set of data distributions. Of the six tasks, scores from the two Easy Solve It tasks (#1 and #4) violated assumptions of normality and homogeneity. This could potentially be due to the fact that these

distributions showed floor effects. Although the mean differences in scores for these two tasks seem to follow the patterns of the other Solve Its, they will be omitted from statistical analysis. The most difficult task (#2, which involves programmatically coordinating multiple characters) was only administered to first and second grade students, as it assessed a content topic that was not covered in Kindergarten (see Tables 1, 2). This task was analyzed using an independent-samples Mann–Whitney U, or Mann–Whitney test to determine differences in the scores of first and second grade students.

General trends in Solve It tasks

Next, we addressed the first research question by exploring children's performance on all Solve It tasks and interpreting for evidence of their logic strategies. We began by examining the Block Recognition Solve It tasks.

Overall, Kindergarteners showed the highest number of average errors per question ($M = 3$), followed by first graders ($M = 1.8$) and second graders ($M = 1.4$). Standard deviation was broader for the Kindergarten class ($SD = .87$) and narrowed in first grade ($SD = .56$), but widened slightly in second grade ($SD = .59$) (see Table 4). These trends indicated that the higher a child's grade in school, the more likely that their performance on the Block Recognition Solve Its would have fewer errors, and that a prediction of their score would be precise.

Researchers investigated whether a student's grade affected their overall performance on Block Recognition Solve Its. A Pearson product-moment correlation was run to determine the relationship between a student's grade and their mean score on the Block Recognition Solve It assessment. There was a strong negative correlation between age and errors made, which was statistically significant in a two-tailed significance test ($r = -.68$, $N = 52$, $p < .001$) (see Fig. 6). To further investigate the relationship, a Pearson product-moment correlation was run on children's age (in years and months) at the time of the assessment against the child's average number of errors. This correlation was even more pronounced ($r = -.71$, $N = 53$, $p < .001$) with younger age very strongly correlated with more mistakes. This indicates that older children in our sample were statistically less likely to make errors than younger ones, by a factor of about 1.3 fewer errors on average for every one-year increase in age (e.g. a six-year-old in our study was likely to make about 2–3 more errors than an eight-year-old on a full six-question Solve Its Assessment).

A non-parametric Kruskal–Wallis H, or Kruskal–Wallis, test was conducted to compare the effect of a student's grade on their performance on the Block Recognition Solve Its in the Kindergarten, first grade, and second grade conditions. There was a statistically significant difference in the number of Block Recognition errors made by Kindergarteners ($Mdn = 3.00$), first graders ($Mdn = 1.67$), and second graders ($Mdn = 1.33$), $H(2) = 27.895$, $p < .001$. However, pairwise comparisons with adjusted p-values revealed

Table 4 Summary statistics for mean scores on ScratchJr block recognition assessment, by grade

	N	Min	Max	*M*	*SD*
Kindergarten	20	1.80	4.80	2.00	0.87
First grade	15	1.00	1.00	1.78	0.56
Second grade	19	0.67	2.67	1.45	0.59

that first grade and second grade mean scores did not significantly differ from each other, $U = 6.184$, *ns,* $r = .195$.

To further elucidate the differences in mean scores across the three grades, Kruskal–Wallis tests were also conducted on each individual Solve It task. Tests revealed a significant main effect of grade on scores for Solve It Tasks #3, $H(2) = 12.969$, $p = .002$, #5, $H(2) = 19.138$, $p < .001$, and #6, $H(2) = 28.145$, $p < .001$. Pairwise comparisons using adjusted p-values showed that for Task #3 showed that Kindergarteners made significantly more mistakes than first, $U = 17.758$, $p = .002$, $r = .58$, and second graders, $U = 12.483$, $p = .031$, $r = 41$. Kindergarteners also made more errors on Task #5 compared to first $U = 17.150$, $p = .003$, $r = .56$, and second graders, $U = 19.855$, $p < .001$, $r = .65$, as well as on Task #6 compared to first $U = 17.400$, $p = .003$, $r = .56$, and second graders, $U = 25.484$, $p < .001$, $r = .83$. As previously mentioned, the most difficult Solve It Task (#2) was only administered to first and second grade students. Although there was no significant difference between second and first grade errors for any other Solve It task, a Mann–Whitney test revealed that the first graders ($Mdn = 2$) made significantly more errors than second graders ($Mdn = 1$) on Solve It tTsk #2, $U = 66.0$, $p = .007$, $r = –.48$.

When it came to Block Sequencing tasks, children's responses were extremely diverse across the sample. Comparative analysis methods were used to determine emergent patterns. Researchers looked at children's Block Sequencing responses in relation to their Block Recognition responses for Solve It Tasks #4, 5, and 6 (see Figs. 7, 8, 9).

In Solve It Task 4, children needed to understand a specific block, Go to Page, that "turns the page" to a different scene. The block that caused the change contained a thumbnail image of the new scene, a concrete reference to the block's action. When children were asked to select the block that could have changed the scene, only 52% of Kindergarteners were able to select the correct block, compared with 100% of first graders and 85% of second graders. When building a sequence of blocks, 76% of Kindergarteners and 90% of second graders added the Go to Page block to their block sequence. First graders maintained their high performance, with 100% including the Go to Page block somewhere in their program.

Solve It Task 5 assessed a child's ability to recognize the Start on Bump trigger block. This block initiates a code when two characters on screen touch. This could have appeared visually chaotic, since one character was already moving toward the target, and the target character then initiated its own movements when they touched. 67% of first graders and 68% of second graders correctly selected the Start on Bump block on the Block Recognition task. On the Block Selection task, 53% of first graders and 40% of second graders included a Start on Bump block in their response.

Solve It Task 6 required children to identify the Speed block, a block that changed the speed of a character's program. Only 5% of Kindergarteners circled a Speed block in their assessment, compared to 25% of first graders, and 45% of second graders. About the same proportion of Kindergarteners (5%) and first graders (27%) used the Speed block in their sequenced code, but the proportion of second graders who used the correct block jumped to 65% of the cohort.

Common Errors in Solve It tasks

One interesting exception to the general mastery of motion blocks is the case of the Hop block in Task #3. Many children responded that the action was created with a Move Up and a Move Down block. A Hop in ScratchJr looks visually different from an Up and

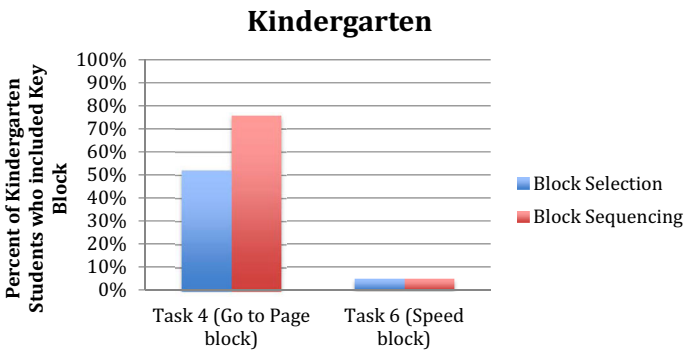


Fig. 7 Bar chart showing differences in block recognition and block sequencing tasks for Kindergarteners

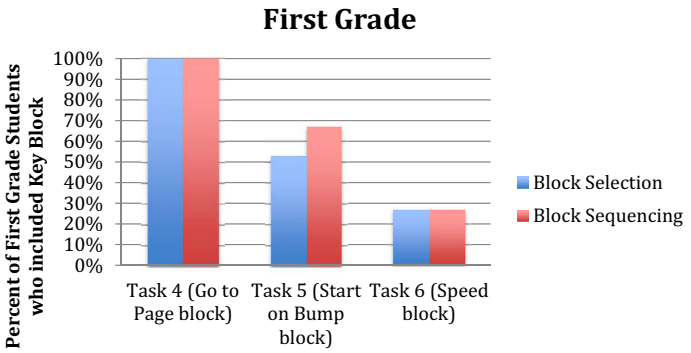


Fig. 8 Bar chart showing differences in block recognition and block sequencing tasks for first graders

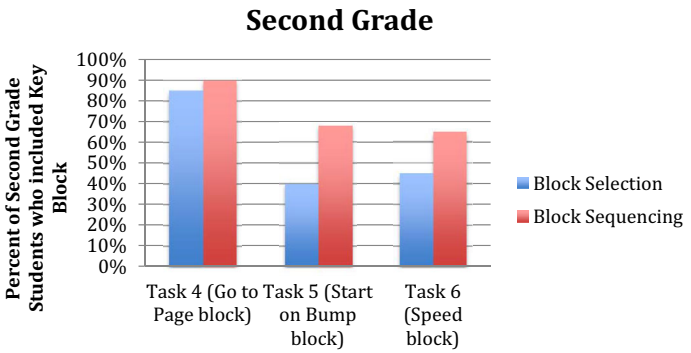


Fig. 9 Bar chart showing differences in block recognition and block sequencing tasks for second graders

Down action, but it is true that any hop can be described as having constituent up and down motions. In order to explore this further, children's responses were compared to Solve It Task 1, which shows a character moving Up and Down. In the majority of cases, children chose one representation, either Up and Down blocks or a Hop block, and applied it to both Task 1 and Task 3. It was less common for children to select different blocks for each question. The proportion of children in each grade who correctly used Hop in both their Block Recognition and Block Sequencing responses was 15% in Kindergarten, to 31% in first grade and 35% in second grade.

As mentioned previously, Solve It Task 2 relied on an advanced concept that was not covered in Kindergarten. This Solve It involved coordinated characters, meaning that two characters on screen took turns performing actions. This could be accomplished in two ways. The first used the Send and Receive Message blocks. These blocks are uniquely dependent on causal reasoning, because they must be coordinated in order to work. The second way to coordinate their actions was to use the Wait block, which uses parallel programming rather than cause-and-effect logic. 13% of first graders selected either a Wait block or a single Message block, compared with 40% of second graders who selected either a Wait or Message block solution. Interestingly, of the children who circled Message blocks, none of the first graders correctly circled both the Send and Receive message blocks, but all of the second graders did.

Knowledge domains evident in programming responses

The next sections address the question of which learning domains children exercise when they engage in computer programming using ScratchJr. In this section we address our second research question, "What strategies from existing knowledge domains (e.g. language, visual-spatial, etc.) are evident as contributors to children's programming logic?"

Seeing is Believing: Spatial reasoning and movement blocks.

Often, research into quantitative assessments focuses on concepts that students misunderstood, or where there was diversity in answers. However, especially in early education, it is also important to note concepts that are generally attainable across a wide age range. All children in this study mastered simple Motion commands in ScratchJr, and correctly mapped the arrow icons of programming blocks to their corresponding movements on a ScratchJr screen. Although this sounds simple enough, icon-based thought is still developing in early childhood, with many Kindergarteners still learning how to match letters with their symbolic meanings (Berninger et al. 2002). The authors hypothesize two related reasons for this mastery of Motion blocks across the entire sample. First, the Motion block images in ScratchJr were chosen to very closely correspond with the movements they represent: the Move Right block depicts an arrow pointing to the right, the Move Up block shows an arrow pointing up, the Hop block shows a parabolic arrow, etc. This is a much more direct visual comparison than, say, the abstract symbol "A" to represent a particular vocal sound. Second, the curricular focus on kinesthetic practices around ScratchJr's concrete symbol system may have helped to ingrain the icon meanings in the children's thinking (Lakoff and Núñez 2000). In-class games and activities like "Simon Says" with programming instructions regularly reinforced the meaning of the block icons by requiring children to physically act out the motions with their bodies. Piaget's stage theory cites concreteness of thought as one of the most fundamental, defining characteristics of early learners (Piaget 1953). Recall that in Task #3, many children mistakenly replaced a Hop block with a combination of Move Up and Move

Down blocks. Rather than suggesting errors or misconceptions, this could be evidence of children working to isolate and identify components of movement in order to reverse-engineer a sequence.

In general, the straightforward link between a Motion block image and its on-screen action allowed children to visually see the connection, which was further strengthened when they can physically act out the motion with their own bodies (e.g. stepping left to simulate a Move Left block). This concreteness allowed children to master simple ScratchJr iconography, even while the youngest children were developing mastery over more familiar symbol systems like the alphabet or numbers. However, the results from the Solve It assessments indicate that programming comprehension goes beyond concrete symbol recognition. The next section looks at programming blocks that rely on abstract thinking to interpret and use correctly.

Blocks to Program Other Blocks: Common mistakes in causal reasoning.

The assessment tasks that received the most variable responses were those that targeted Control Flow blocks. Unlike the Motion blocks, Control Flow blocks do not program characters directly, but instead modify how programs are executed. Speed blocks do not make a character run quickly, but rather they make all of that character's programs happen more quickly. Repeat loops will make any action inside of them repeat, but an empty Repeat loop does nothing on its own. In fact, the defining feature of most Control Flow blocks is that they do not make sense, or sometimes do not even function, without other blocks attached. This is because Control Flow blocks do not act on characters, but rather on other blocks.

According to developmental stage theory, children mainly use tactile experiences to understand the world around them, and can be confused by processes that are invisible (Piaget 1953). That is exactly why the research team deems Control Flow tasks "difficult" questions. But the benefit of bringing these invisible processes into a programming language is that children can at least view a concrete instruction list and observe its consequences in the form of the resulting animation before and after the changes. Seymour Papert (1980) realized the power of this visual demonstration to engage children in evaluating logic and thinking processes from a bird's-eye level, and subsequently, to expose children to simple practices of metacognition.

In children's responses, we observed variations in children's ability to engage in causal logic. Two examples of common errors from the Solve Its demonstrate this clearly.

Example 1, Changing the Scene: In Solve It Task 4, recall that Kindergarten and second grade children showed slightly lower performance in selecting a necessary block when compared to first graders, who selected the correct block 100% of the time. It is unclear why the first grade cohort had such a high rate of recognizing this block, even surpassing the second graders, but it might have something to do with overgeneralization. In interviews with young children exploring balance and weight of unfamiliar objects, Karmiloff-Smith and Inhelder (1975) found that "most of those subjects who had been successful in balancing conspicuous or inconspicuous weight blocks because they had been concentrating on the goal, began later in the session to experience serious difficulties in repeating the successful action as their attention shifted to the means" (pg. 203). Although it is unclear what theories-in-action are being tested or exercised when children engage in programming, this pattern of older children underperforming compared to younger might indicate their more reflective approach to solving the problem. Karmiloff-Smith and Inhelder write that "overgeneralization is not only a means to simplify but also to unify […and] can be looked upon as the *creative simplification* of a problem by ignoring some of the complicating factors (such as weight in our study)" (1975, original emphasis). By this

logic, the second graders may be exhibiting some creative simplification, holding one variable in order to more deeply understand another one. More research is needed to understand whether there are similar complicating factors that are being held constant when second grade children work through a programming challenge.

Example 2, Changing a Character's Speed: Solve It Task 6 required children to identify the Speed block as a cause for the changing tempo of the character's action. As reported earlier, researchers found that very few Kindergarteners identified this block (5% on Block Recognition, 5% on Block Sequencing), compared to about a quarter of first graders (25% on Block Recognition, 27% on Block Sequencing), and about half of second graders (45% on Block Recognition, 67% on Block Sequencing). The fact that older children more easily understood the Speed block could mean that they are better able to recognize the cause of a block simply from viewing its result. However, it may also mean that second graders in this study were able to devote more working memory or attention to their initial observation of the program. In other words, the Kindergarteners simply may not have even noticed the change in speed, possibly because of the demanding assessment environment or the mental limits of trying to focus on other elements of the program. Demetriou et al. (2010) describe developmental trajectories in cognitive processing, such that as children grow they become more able to manage, differentiate, and operate on their own perceptions. Their research shows that a dramatic increase in processing functions occurs between first and second grade, and prior to that children demonstrate very preliminary stages of processing capacity. Further, older children can rely on more than one domain (e.g. visual, causal, and spatial) at once rather than attending to only one or two. This raises an interesting question about the interaction of multiple domains as they contribute to children's developing programming knowledge. In the next section, we further interpret these results and extend findings into areas for future research.

Discussion

In summary, students generally performed on Solve It tasks in a manner consistent with what one would expect from the pilot trials of ScratchJr (Flannery et al. 2013). Children at the Kindergarten level had more trouble with meta-level control flow blocks and coordinating multiple characters. First and second grade students found these concepts more accessible, and spent more time exploring complex systems of instructions and multi-step strategies to further develop programmed stories and even games. Again, these concepts map onto other areas of cognitive growth for this slightly older age, such as using and inventing rules for games, modeling problems visually and abstractly, and creating short narratives for different audiences (MA DOE 2011a, b).

At first glance, the results presented above might not seem overly surprising to education researchers or practitioners. It is rather predictable that young children, as they grow older, perform progressively better on assessments of cognition and memory related to school tasks. There are two unique elements in these findings, however.

First, although the trajectory of improvement over successive grades is a familiar pattern, the domain of programming is a relatively new area of study for developmental theorists. It cannot be assumed that current theories of developmental trajectories in traditional academic subjects will apply to all burgeoning learning domains. The fact that these patterns were observed in the context of computer science and engineering education provides an opportunity to describe and explain these trends using theoretical approaches from existing knowledge domains.

The second unique finding in these results is in the nature of the observed programming errors. The findings presented thus far suggest that a variety of pre-existing knowledge domains are leveraged as children develop programming fluency. For example, recall that in Solve It Task 2, only about 13% of first graders and 40% of second graders correctly selected one of the necessary blocks (Wait, Send Message, or Receive Message). No child in the sample correctly selected both of the required Message blocks. This suggests that causal reasoning is key to interpreting the programmatic relationship between ScratchJr characters. First grade children in our sample showed evidence that they still developing their ability to observe and deduce causes for visible cause-and-effect relations. Meanwhile, second graders were learning to develop theories about invisible causal relationships and possibly reason about those relationships using their past experiences (Barrouillet and Gaillard 2010). Spatial knowledge may also play a role in children's ability to solve this task. Children age 5–6 years (roughly first grade aged) are known to rely more on single dimensional representations, where 7–8 year olds (i.e. second graders) are able to demonstrate "fluent" mental imagery (Demetriou et al. 2010). This could explain why second graders tested better when interpreting Solve Its with multiple moving characters at once. Finally, these results may have interesting implications for the integration of verbal logic into early programming experiences. Children in first grade can typically use and apply permission rules in language, meaning they can coherently express thoughts using simple word sequences and grammatical structures. By the time children reach second grade, they can also use inference in communicating ideas. (Barrouillet and Gaillard 2010; Demetriou et al. 2010). They are better able to apply facts and techniques from multiple domains towards solving a problem, without needing to recall all of the theories and examples from those individual domains of knowledge. This implies that the development of domain-specific processes, particularly verbal reasoning, may positively impact a child's ability to perform logic operations relevant to programming, including inference about invisible systems.

This line of reasoning raises the question, if verbal reasoning may play a role in programming comprehension, then how might other domains be involved? This study does not claim to report on children's social or emotional states while working on their ScratchJr projects, but there is evidence that they were deeply engaged in social reasoning throughout the lessons. Field notes from all three classrooms show that students in all cohorts created inventive, story-driven projects, and second graders were often observed making projects with multiple plot stages or game-like elements (i.e. characters and objects programmed to be interactive with a user). Second graders were more likely to share projects with classmates or to work on projects collaboratively, either by sharing an iPad, using one child's project as a template to copy, or constructing multi-project story lines with multiple children's iPads. In addition, they were the only children observed relating classroom content from other units (science and social studies lessons, etc.) into their collages, stories, and games (see Figs. 10, 11). This description calls to mind what Wyeth (2008) refers to as "purposeful, task oriented programming," or programming with a goal, which is more sophisticated than simply building a program to solve a problem. It is possible that the development of social reasoning, which involves deciphering emotions, interpreting intentions of others, developing relationships, and imagining alternative perspectives, may all contribute to, or be correlated with, programming logic and practices.

Fig. 10 This group of second graders spontaneously chose to work together on a "fashion show" project, with each child's ScratchJr project representing a similar but different portion of the show

Fig. 11 This second grade girl built a project about migration, using a picture of a map that hangs in her classroom as the background of her animation. Maps and migration are concepts covered in the class Social Studies unit. Other second graders incorporated moon phases, a topic from their Science unit, into their games and stories

Interestingly, the Solve It assessments did not capture evidence of categorical or quantitative reasoning, two knowledge domains commonly associated with computer programming (Confrey 1990; Robins et al. 2003). It is possible that quantitative skills (e.g. subitization, counting, pointing, adding) and categorical ones (e.g. noting similarity and difference, construction of mental categories for organization) may not be relevant to core programming skills of sequencing and inferring cause-and-effect relations (Barrouillet and Gaillard 2010). It is likely that these developing domains play indirect roles in shaping a child's programming processes. In order to observe the nature of programming comprehension development in young children, it seems necessary to begin comparison to other long-standing knowledge domains, such as those suggested by Demetrious et al. (2010), where more is known about children's representations and patterns of development.

Limitations

In the course of implementing this programming intervention, there were several challenges to data collection and interpretation. The major limitation was sample size. However, given the challenges of naturalistic classroom action research in the early childhood context, the data pool from three classrooms was actually quite large and rich. Field notes from this study raised questions about collaboration and intentionality in children's programming, especially among older children in the sample. These findings have inspired new studies within our research group, to explore collaboration and internal motivation in second graders exploring ScratchJr (Portelance et al. 2015).

The assessment format also presented interesting complexities. Based on the paper-and-pencil method of administering the Solve It tasks, it was not possible to control for the contextual difference between programming with ScratchJr on a touchscreen tablet and manipulating paper cut-outs of programming blocks. Interestingly, using this format allowed children to express things not possible on the ScratchJr iPad app such as sequencing upside-down and sideways blocks, leaving different sized gaps between blocks (sometimes in place of Wait blocks, ostensibly to indicate a pause in the program), and mixing character icons with programming blocks to indicate when a character should move. These alternative answers were not analyzed here as they did not pertain to directly to ScratchJr programming knowledge, but they showed creativity in problem-solving and exploration. Future work with the Solve It assessments will specifically examine creativity in programming development.

Finally, although the results presented here suggest that younger children may be developmentally unable to master difficult ScratchJr concepts to the same degree as older children, it is important to remember that children in all classes were given the same amount of ScratchJr lesson time. In typical early childhood settings, younger children spend more time on curricular units relative to older ones (Kostelnik and Grady 2009). It is possible that, given more time to practice and explore the ScratchJr interface, Kindergarten children could master programming concepts as thoroughly as older students. Future work should examine children's developmental capacity to explore programming concepts, and describe lengths of time required in each range to achieve such mastery.

Implications and future work

The main finding from this study is that developmental level impacts children's emerging programming knowledge, which has major implications for the field of cognitive development. Our society has recently seen a massive shift in cultural attitudes towards computer coding, with a booming global market for home-use educational technologies, and state and national curricula in the U.S. and abroad now mandating computer science education (Balanskat and Engelhardt 2015; K–12 Computer Science Framework 2016; Livingstone 2012). However, this study suggests that the presumed educational benefits of coding, such as math skills and problem solving, may not be as evident in early childhood as other learning domains, such as verbal, causal, and social reasoning. In the context of cognitive developmental stage theory, these findings also suggest that there is a developmental progression to the way children learn programming knowledge. Piaget might argue that all of the diversity in children's responses are simply symptomatic of their different points along their general intellectual developmental trajectory (Piaget 1953; Flannery and Bers 2013). However, developmental theorist Feldman would argue that programming may constitute a unique learning domain, less widely-applicable than universal developmental milestones such as walking and talking, but characterized by developmental growth nonetheless (Feldman 1980). In his non-universal theory, he has argued that the key question for determining the nature of a domain of knowledge is to ask whether the domain can be organized into a sequence of developmental levels, and then to operationalize the specificity of that knowledge. The evidence from this study suggests that such a sequence of developmental levels in programming knowledge does, in fact, exist. Future work into the nature of programming knowledge should seek to further understand this progression. Researchers should explore children's learning using a broad array of educational programming technologies, and should draw on education and design movements, such as computational thinking with low- or no-tech activities (Bers 2018).

Additionally, this study provides preliminary evidence that children's programming knowledge may draw on various specialized knowledge domains. Prior work has established that knowledge domains may follow their own developmental trajectories, although they are inherently interconnected (Barrouillet and Gaillard 2010; Demetriou et al. 2010). If early programming learning is rooted in dynamic relations among these emerging domains, then future work should concentrate on understanding the developmental progressions of children's programming knowledge. Perhaps through this work, we can arrive at a developmental trajectory of programming learning to better inform research, design, and education fields.

Practically, operationalizing a programming domain into developmental levels would be a foundational step for creating evidence-based computer science curricula. Currently, many frameworks and guidelines exist, along with mandates from state and national governments to teach computer science concepts (Balanskat and Engelhardt 2015; K–12 Computer Science Framework 2016; Livingstone 2012). Frameworks should consider the developmental progression of programming knowledge, and should incorporate current findings to strengthen their applicability to a wider audience of children. Additionally, designers can take a developmental stance when building new coding technologies for early childhood. Development levels of programming knowledge should inform tool and curriculum design such that young learners are scaffolded through a progression of one or many developmental levels while the same programming environment.

Conclusion

Computer programming for young children has become a flourishing area for education research, and this study contributes to the investigation of cognitive capabilities of young students to learn computer programming. This study shared results from a programming assessment for early childhood (K-2nd grade), and used common errors on the assessment to argue for the cross-domain nature of computer programming knowledge. Children in our sample were able to observe a programmed animation and deduce, through reverse logic, the programming instructions necessary to create the animation. The varied performance of Kindergarteners, first graders, and second graders, the age-specific common errors, and the lack of gender-based differences all suggest that cognitive developmental level is the major factor to consider when researching young children's programming knowledge development.

Contrary to popular opinion about the mathematical nature of programming knowledge, this study suggests that causal, spatial, verbal, and social reasoning all play a key role in children's programming learning. Excepting causal reasoning, which involves logical reasoning, the other domains are not often leveraged in the design of programming tools and interventions for children in early childhood. Practitioners can take this finding as a justification for interdisciplinary and project-based programming experiences in early childhood learning settings, in which children can pursue story-based, collaborative, and physical explorations of coding. Designers of educational technology should consider this a call to action, to incorporate features that can support these developing domains. For example, "multiplayer" programming environments can tap into social reasoning, 3D and augmented-reality programming platforms support spatial experiences, and programming languages that leverage the syntax of spoken languages can engage children's verbal reasoning skills.

Most importantly, findings from this study hold implications for the importance of introducing programming and coding at a very young age. Children in our study demonstrated a range of mastery and creativity through coding, but none showed a complete lack of understanding about the most foundational programming concepts after the intervention. This supports the growing body of research about the benefits of computer programming to support the development of other cognitive domains (Bers 2012, 2018; Clements and Samara 2003). The new medium of programming is quickly becoming a new literacy, which children can use to connect with the logic of algorithms, or the expressive possibilities of digital storytelling (Bers 2018). As with all literacies, programming now deserves rigorous empirical investigation, evidence-based practice recommendations, and a policy-level response to the cultural mandate that programming is now a general knowledge requirement for children from Kindergarten to 12th grade and beyond.

Acknowledgements This work was generously funded by the National Science Foundation Grant No. DRL-1118664.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Appendix A: Answer key for Solve It tasks

ScratchJr Solve-It Answer Sheet

Name of Student:

FIX THE PROGRAM

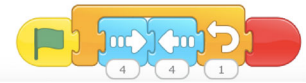
Question 1: Remove a block

Circle a block to remove

Question 2: Add a block

Circle a block to add:

Question 3: Modify blocks



Appendix B: Scoring rubric for Solve It tasks

ScratchJr Solve-Its scoring rubric: sets of blocks

For every question, there is a bank of correct answers. Responses will be scored based on how close they are to some correct answer from that bank.

1. How scoring will be assessed
    a. A score of 0 is perfect
    b. If there is a correct block that is missed, that is 1 point
    c. If there is an incorrect block that was added, that is 1 point
2. Instances with responses that are "more correct" based on block similarity
    a. There is a key of "similar" blocks. If a student missed a correct block and used an incorrect *but similar* block, that is 1 point (not 2)
3. Instances with more than 1 correct answer
    a. There is a key of correct block sets for all questions. The response will be compared against every set. The lowest of these possible scores will be counted.
    b. Ex: Correct Program A = Green Flag, Show, Hide, Message Block
    Correct Program B = Green Flag, Show, Hide, Wait Block
    Submitted Response = Green flag, Show, Hide, Stop

Score and Justification:

The submitted response will receive a score of 1 based on Program B, because the Stop block is similar to the Wait block.

They would have received a score of 2 based on Program A, because the Message Block was missing, and the non-similar Stop block was added.

Similar blocks key

Jump replaced by Up + Down (1 point)
Grow + Shrink replace by Hide + Show (2 points)
Added End block or Go Home at end of program when it would make no change (0 points)
Used go home in place of last block when it is incorrect, but the character does end up "at home", i.e. in Sol2 (1 point)
No Response = total points missed. Do not scrap this data, could mean child did not understand the question
Move Right replaced by Go Home in Q6 (gave 2 points)

ScratchJr Solve-Its scoring rubric: sequences of blocks

For every question, there is a bank of correct answers. Responses will be scored based on how close they are to any correct answer from that bank.

1. How scoring will be assessed
    a. A score of 0 is perfect

b. All rules of "set" scoring still apply
c. Sequences will be broken into "chunks" of blocks that represent different functions of a program (i.e. begin and ending chunks, action chunks, control-flow chunks)

2. Scoring Chunks

a. If the blocks that comprise a chunk are incorrectly ordered, that is 1 point
b. If the chunk is incorrectly ordered relative to other chunks, that is 1 point
c. If there is a chunk that is missing, that is 1 point
d. If there is a chunk that is incorrectly added, that is 1 point

3. Instances with incorrect sets of blocks

a. For blocks that are incorrect but similar to the correct block (according to similarity key), those blocks count towards the chunk that the correct block would have filled.
b. For blocks that are incorrect and dissimilar from the correct block, they are counted as an incorrectly added chunk.

4. General Rules of Chunks:

a. Begin blocks need to be at the beginning
b. End blocks need to be at the end
c. Chunks with more than one block need to have blocks in the correct order (1 point if wrong) and need to be adjacent (1 point is separated)
d. Control Flow block must operate on a block to its right (1 point if there no motion block after it)
e. Character blocks need to be associated with a single program (anywhere inside or very near is fine). If characters are in the incorrect program, or if they are not associated with a program, that is 1 point.

1. To determine if the characters are "flipped," score each program for each character. Whichever yields the lower score is the pair of response that is accepted. If the character blocks are incorrectly matched based on the accepted response, that is 1 point.

- Question 4b: Rules of Chunks
  - Start Chunk
    - Start block must go at the beginning of the program (1 point if in incorrect location)
  - Action Chunk
    - 2 action blocks must be adjacent (1 point if not adjacent)
    - 2 action blocks must be in correct order (1 point if incorrectly ordered)

- End Chunk
  - End block must go at the end of the program (1 point if in incorrect location)
  - End block must be Go To Page 2 (1 point if similar end blocks is used instead)

- Question 5b: Rules of Chunks
  - Cat Program – Start Chunk
    - Start block must go at the beginning of the program (1 point if in incorrect location)
  - Cat Program—End Chunk
    - End block must go at the end of the program (1 point if in incorrect location)
    - *Special block: If there is no End block, that is 2 points*. See Rubric Justifications.
  - Pig Program—Start Chunk
    - Start block must go at the beginning of the program (1 point if in incorrect location)
    - *Special Block: If there is no Start block, that is 2 points*. See Rubric Justifications.
  - Pig Program – Action Chunk
    - 2 action blocks must be adjacent (1 point if not adjacent)
    - 2 action blocks must be in correct order (1 point if incorrectly ordered)
  - Character Chunk
    - Character chunks must be associated with (adjacent or very near) one of the programs (1 point each for missing characters)
    - Ambiguous Character-Program Pairs:
      - Score each program for each character. Whichever yields the lower score is the pair of response that is accepted (Ex: Cat + A = 12, Cat + B = 5; Pig is same for both. B is Cat's program, A is Pig's program). If the character blocks are incorrectly matched based on the accepted response, that is 1 point.

- Question 6b: Rules of Chunks
  - Start Chunk
    - Start block must go at the beginning of the program (1 point if in incorrect location)
  - Action Chunk
    - 3 action blocks must be adjacent (1 point if not adjacent)
    - 3 action blocks must be in correct order (1 point if incorrectly ordered)

- Speed block must operate on a block to its right (1 point if there no motion block after Speed in this chunk)
- *Special Block: If there is no Speed block, that is 2 points*. See Rubric Justifications

- End Chunk
  - End block must go at the end of the program (1 point if in incorrect location)
    - NOTE: in this program, the "Go Home" block is functionally the End block. No similar blocks will be accepted if missing, although adding an End block will not affect set or sequence score.

References

Balanskat, A., & Engelhardt, K. (2015). *Computing our future computer programming and coding-priorities, school curricula and initiatives across Europe*. Brussels: European Schoolnet.

Barrouillet, P., & Gaillard, V. (Eds.). (2010). *Cognitive development and working memory: A dialogue between neo-Piagetian theories and cognitive approaches*. Hove: Psychology Press.

Berninger, V. W., Abbott, R. D., Vermeulen, K., Ogier, S., Brooksher, R., Zook, D., et al. (2002). Comparison of faster and slower responders to early intervention in reading: Differentiating features of their language profiles. *Learning Disability Quarterly, 25*(1), 59–76.

Bers, M. U. (2008). *Blocks to robots: Learning with technology in the early childhood classroom*. New York, NY: Teachers College Press.

Bers, M. U. (2012). *Designing digital experiences for positive youth development: From playpen to playground*. Cary, NC: Oxford.

Bers, M. U. (2014). Tangible kindergarten: Learning how to program robots in early childhood. In C. I. Sender (Ed.), *The go-to guide for engineering curricula PreK-5: Choosing and using the best instructional materials for your students* (pp. 133–145). Thousand Oaks, CA: Corwin.

Bers, M. U. (2018). *Coding as a playground: Programming and computational thinking in the early childhood classroom*. New York: Routledge Press.

Case, R. (1992). *The mind's staircase: Exploring the conceptual underpinnings of children's thought and knowledge*. Hillsdale, NJ: Erlbaum.

Case, R., Demetriou, A., Platsidou, M., & Kazi, S. (2001). Integrating concepts and tests of intelligence from the differential and developmental traditions. *Intelligence, 29*(4), 307–336.

Clements, D. H. (2002). Computers in early childhood mathematics. *Contemporary Issues in Early Childhood, 3*(2), 160–181.

Clements, D. H., & Sarama, J. (2003). Strip mining for gold: Research and policy in educational technology-A response to "Fool's Gold". *Educational Technology Review*, *11*(1), 7–69. Retrieved from https://www.researchgate.net/profile/Douglas_Clements/publication/228557118_Strip_mining_for_gold_Research_and_policy_in_educational_technology-A_response_to_Fool's_Gold/links/0c960529762a5e538f000000.pdf

Confrey, J. (1990). A review of the research on student conceptions in mathematics, science, and programming. *Review of Research in Education, 16*, 3–56. Retrieved from http://www.jstor.org/stable/1167350

Corbin, J., & Strauss, A. (2008). *Basics of qualitative research: Techniques and procedures for developing grounded theory* (3rd ed.). Thousand Oaks, CA: Sage.

Davies, S. P. (1993). The structure and content of programming knowledge: Disentangling training and language effects in theories of skill development. *International Journal of Human-Computer Interaction, 5*(4), 325–346.

Demetriou, A. (2000). Organization and development of self-understanding and self-regulation: Toward a general theory. In M. Boekaerts, P. R. Pintrich, & M. Zeidner (Eds.), *Handbook of self-regulation* (pp. 209–251). Washington: Academic Press.

Demetriou, A., Spanoudis, G., & Mouyi, A. (2010). A three-level model of the developing mind: Functional and neuronal substantiation and educational implications. In M. Ferrari & L. Vuletic (Eds.),

Developmental relations among mind, brain and education. Dordrecht: Springer. https://doi.org/10.1007/978-90-481-3666-7_2.
Elkind, D. (1961). Children's discovery of the conservation of mass, weight, and volume: Piaget replication study II. *The Journal of Genetic Psychology, 98*(2), 219–227.
Feldman, D. H. (1988). Universal to unique: Toward a cultural genetic epistemology. *Archives de Psychologie, 56*(219), 271–279.
Feldman, D. H. (2004). Piaget's stages: The unfinished symphony of cognitive development. *New Ideas in Psychology, 22,* 175–231. https://doi.org/10.1016/j.newideapsych.2004.11.005.
Fischer, K. W. (1980). A theory of cognitive development: The control and construction of hierarchies of skills. *Psychological Review, 87,* 477–531.
Flannery, L. P., & Bers., M. U. (2013). Let's dance the "robot hokey-pokey!": children's programming approaches and achievement throughout early cognitive development. *Journal of Research on Technology in Education, 46*(1), 81–101. http://ase.tufts.edu/DevTech/publications/JRTE-robot-hokey-pokey.pdf
Flannery, L.P., Kazakoff, E.R., Bontá, P., Silverman, B., Bers, M.U., & Resnick, M. (2013). Designing ScratchJr: Support for early childhood learning through computer programming. In *Proceedings of the 12th international conference on interaction design and children* (*IDC '13*) (pp. 1–10). ACM, New York, NY, USA. https://doi.org/10.1145/2485760.2485785\
Fletcher-Flinn, C. M., & Gravatt, B. (1995). The efficacy of computer assisted instruction (CAI): A meta-analysis. *Journal of Educational Computing Research*, *12*(3), 219–241.
K–12 Computer Science Framework. (2016). Retrieved from http://www.k12cs.org
Kafai, Y. B., & Resnick, M. (1996). *Constructionism in practice: Designing, thinking, and learning in a digital world*. New Delhi: Routledge.
Karmiloff-Smith, A., & Inhelder, B. (1975). If you want to get ahead, get a theory. *Cognition, 3*(3), 195–212.
Kazakoff, E. R. (2014). Cats in Space, Pigs that Race: Does self-regulation play a role when kindergartners learn to code? (Unpublished doctoral dissertation). Tufts University, Medford, MA. Retrieved from: http://ase.tufts.edu/DevTech/resources/Theses/EKazakoff_2014.pdf
Kazakoff, E., & Bers, M. (2012). Programming in a robotics context in the kindergarten classroom: The impact on sequencing skills. *Journal of Educational Multimedia and Hypermedia, 21*(4), 371–391. Retrieved from https://ase.tufts.edu/DevTech/publications/JEMH.pdf
Kazakoff, E., Sullivan, A., & Bers, M. U. (2013). The effect of a classroom-based intensive robotics and programming workshop on sequencing ability in early childhood. *Early Childhood Education Journal, 41*(4), 245–255. https://doi.org/10.1007/s10643-012-0554-5.
Kostelnik, M. J., & Grady, M. L. (2009). *Getting it right from the start: The principal s guide to early childhood education*. Thousand Oaks, CA: Corwin Press.
Lakoff, G., & Núñez, R. (2000). *Where mathematics comes from: How the embodied mind brings mathematics into being*. New York: Basic Books.
Lightfoot, C., Cole, M., & Cole, S. (Eds.). (2009). *The development of children* (6th ed.). New York: Worth.
Livingstone, S. (2012). Critical reflections on the benefits of ICT in education. *Oxford Review of Education, 38*(1), 9–24.
MA DOE. (2011a). Massachusetts curriculum framework for English language arts and literacy. Retrieved from http://www.doe.mass.edu/frameworks/ela/0311.pdf
MA DOE. (2011b). Massachusetts Curriculum Framework for Mathematics. Retrieved from http://www.doe.mass.edu/frameworks/math/0311.pdf
McDevitt, T. M., & Ormrod, J. E. (2002). *Child development and education*. Upper Saddle River, NJ: Merrill/Prentice Hall.
Mioduser, D., Levy, S. T., & Talis, V. (2009). Episodes to scripts to rules: Concrete-abstractions in kindergarten children's explanations of a robot's behavior. *International Journal of Technology and Design Education, 19*(1), 15–36.
Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books.
Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology, 2,* 137–168.
Piaget, J. (1953). *The origins of intelligence in the child*. London: Routledge and Kegan Paul.
Portelance, D. J., Strawhacker, A., & Bers, M. U. (2015). Constructing the ScratchJr programming language in the early childhood classroom. *International Journal of Technology and Design Education*. https://doi.org/10.1007/s10798-015-9325-0.
Resnick, M. (2006). Computer as paintbrush: Technology, play, and the creative society. In D. Singer, R. Golikoff, & K. Hirsh-Pasek (eds.), *Play = Learning: How play motivates and enhances children's cognitive and social-emotional growth*. Oxford: Oxford University Press

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education, 13*(2), 137–172.
Strawhacker, A. L., & Bers, M. U. (2015). "I want my robot to look for food": Comparing children's programming comprehension using tangible, graphical, and hybrid user interfaces. *International Journal of Technology and Design Education*. https://doi.org/10.1007/s10798-014-9287-7.
Sullivan, A., & Bers, M. U. (2013). Gender differences in kindergarteners' robotics and programming achievement. *International Journal of Technology and Design Education, 23*(3), 691–702.
Vizner, M. (2017). Big Robots for Little Kids: Investigating the role of scale in early childhood robotics kits (Unpublished master's thesis). Tufts University, Medford, MA.
Wyeth, P. (2008). How young children learn to program with sensor, action, and logic blocks. *Journal of the Learning Sciences, 17*(4), 517–550. https://doi.org/10.1080/10508400802395069.

Amanda Strawhacker is a doctoral student at the Developmental Technologies Research group at Tufts University. Her research interests include designing and researching developmentally appropriate technologies and technology-rich spaces for early childhood.

Marina Umaschi Bers is a professor at the Eliot-Pearson Department of Child Study and Human Development and the Computer Science Department at Tufts University. She heads the interdisciplinary Developmental Technologies research group which focuses on developing, implementing, and evaluating innovative learning technologies to promote positive technological development.