J. Comput. Educ. (2019) 6(4):499–528
https://doi.org/10.1007/s40692-019-00147-3

Coding as another language: a pedagogical approach for teaching computer science in early childhood

Marina Umaschi Bers[1]

Received: 28 February 2019 / Revised: 18 June 2019 / Accepted: 9 September 2019 /
Published online: 25 September 2019

Abstract Computer programming is an essential skill in the 21st century and new policies and frameworks aim at preparing students for computer science-related professions. Today, the development of new interfaces and block-programming languages facilitates the teaching of coding and computational thinking starting in kindergarten. However, as new programming languages that are developmentally appropriate emerge, there is a need to explicitly conceptualize pedagogical approaches for teaching computer science in the early years that embrace the maturational stages of children by inviting play and discovery, socialization, and creativity. Thus, it is not enough to copy models developed for older children, which mostly grew out of traditional Science, Technology, Engineering and Math (STEM) disciplines and instructional practices. This paper describes a pedagogical approach for early childhood computer science called "Coding as Another Language" (CAL), as well as six coding stages, or learning trajectories, that young children go through when exposed to CAL curriculum. CAL is grounded on the principle that learning to program involves learning how to use a new language (a symbolic system of representation) for communicative and expressive functions. This paper proposes that, due to the critical foundational role of language and literacy in the early years, the teaching of computer science can be augmented by models of literacy instruction. CAL supports young children in transitioning through different six coding stages. Case studies of young children using either the KIBO robot or the ScratchJr app will be used to characterize each coding stage and to illustrate the instructional practices of CAL curriculum.

Keywords Coding · Young children · Early childhood · Literacy

✉ Marina Umaschi Bers
Marina.bers@tufts.edu

[1] Tufts University, 105 College Ave, Medford, MA 02155, USA

Introduction: new ways of thinking through computer science

Computer programming is becoming an essential skill in the 21st century. Each month, there are an estimated 500,000 openings for computing jobs nationwide, and a lack of adequately trained people to fill them (Code.org 2018; Fayer et al. 2017). In order to meet the growing needs, new educational policies and frameworks aim at preparing students in kindergarten to high school for computer science-related professions (National Research Council 2012; Hubwieser et al. 2014).

This paper addresses this need but claims that the rationale for supporting the introduction of computer science starting in kindergarten shouldn't be the creation of the future workforce, but the future citizenry. If we do not understand what an algorithm is, we might not understand why and how certain information is or is not presented to us. We become illiterate in the information age. Coding is a new literacy, and as such, those who learn how to code from a young age will not only be able to participate in the automated economy, but will also have a civic voice. Reading and writing, as well as computer programming, are tools of power because they support new ways of thinking and the making of new processes and artifacts. From smart watches to cell phones to automated cars, most of our objects have been programmed. Furthermore, algorithms dictate the news displayed in our social media, the people we might enjoy meeting, and the merchandise we might want to purchase.

Researchers have coined the term "computational thinking" to refer to the analytical process rooted in the discipline of computer science and the activity of programming. It involves thinking recursively, applying abstraction, breaking up a complex problem in smaller tasks, and using heuristic reasoning to discover a solution (Wing 2006a, b; 2011). There is debate among researchers and educators regarding whether computational thinking can be classified as a unique category of thought (Gadanidis 2017; Pei et al. 2018). However, the term has grown popular at a time when schools are starting to incorporate computer science in more massive ways (K–12 Computer Science Framework Steering Committee 2016).

While back in the 1960s it was argued that all college students needed to learn programming and the "theory of computation" (Guzdial 2008; Grover and Pea 2013; Perlis 1962), most of its early application was primarily found in mathematics, science and engineering. As new programming languages were developed, such as Basic and Pascal, computer programming started to slowly make its way into high schools and middle schools (Wilson et al. 2010). In the 1980s with the widespread use of LOGO and the turtle that could draw geometrical shapes with simple commands, programming also received a major push in elementary schools (Papert 1980; Abelson and DiSessa 1981). Today, with the development of new computer interfaces and block-programming languages, coding is also arriving to early childhood education. For example, the ScratchJr programming app and the KIBO robotics kit (for a depth discussion of the theoretically rooted design of these tools, see Bers 2018a). Each of these tools engages children to create their own projects and express their ideas through programming. However, research shows that the best uses happen when the tools are integrated into an intentional learning experience and curriculum (Bers 2012; Pea and Kurland 1984) that must be age-appropriate.

Coding in early childhood

Research shows the economic and developmental impact of educational interventions that begin in early childhood. These are associated with lower costs and more durable effects than interventions that begin later on (e.g., Cunha and Heckman 2007; Heckman and Masterov 2007; National Research Council Committee on Early Childhood Pedagogy 2001; Shonkoff et al. 2000). Although there are no comprehensive longitudinal studies yet on the impact of teaching computer science in the early years, it is expected that results will be similar to other areas, given the plasticity of young children. Thus, most states and non-profits, when conceiving standards and programs, start in kindergarten, and sometimes pre-kindergarten.

Widespread organizations such as Code.org, free programming apps such as ScratchJr (Bers and Resnick 2015), and robotic kits such as KIBO (Bers 2018a) provide developmentally appropriate platforms for young children. However, technology and pedagogy are not the same thing. What are the best pedagogical approaches for teaching computer science in the early years? This paper proposes that, due to the critical foundational role of language and literacy in the early years, traditional approaches can be augmented by models of literacy instruction. It is not enough to copy models designed for older children, which mostly grew out of traditional STEM disciplines and instructional practices. In early childhood, approaches must be consistent with developmentally appropriate practice (Bredekamp 1987) and must embrace the maturational stages of children by inviting play and discovery, socialization, and creativity (Bers 2018a).

This paper describes a pedagogical approach for early childhood computer science called "Coding as Another Language" (CAL), as well as six coding stages that children go through as they engage with the CAL curriculum. CAL is grounded on the central principle that learning to program involves learning how to use a new language (a symbolic system of representation) for communicative and expressive functions. First, the paper will present the traditional "Coding as STEM" model. Then, the novel "Coding as Another Language" (CAL) approach will be introduced. Following, six different coding stages that young children move through when learning to code with a CAL approach will be introduced. When presenting these stages, similarities to and differences from the stages of literacy instruction as described by researchers such as Chall (1983) and others (Ryan 2011; Clarke et al. 2015) will be discussed. Case studies of young children using either the KIBO robot or the ScratchJr app will be used to characterize each learning stage and to illustrate the instructional practices of CAL curriculum. Finally, the conclusion will identify guiding principles for CAL.

The traditional perspective: coding as STEM

The STEM acronym came into the American consciousness in the 1950s as a response to the need for a technically oriented workforce, and to maintain national security. In 1958, during the height of the space race, the United States passed the National Defense Education Act (NDEA), which provided funding and incentives for schools to improve their math, science, and engineering curricula to prepare the future workforce. The act also had provisions for research and experimentation in the use of television, radio, and motion pictures for educational purposes and included the teaching of modern foreign languages due to their importance for national security.

As the cold war ended, the emphasis on national security diminished and the perceived urgency to teach a foreign language dropped, but the need for economic competitiveness remained. With a rapidly growing technological society, learning computer programming provided increased career opportunities. However, computer programming was seen as a skillset for mathematicians, scientists, and engineers. Thus, the teaching of computer science drew from methodologies already used in STEM disciplines such as solving pre-set challenges and engaging in competitions. At the time, the broader benefits for everyone to learn how to code could not yet be perceived, as computers did not play a major role in everyday life.

As technological advances rapidly grew and a gender and racial gap started to emerge among STEM-related fields, it became clear that computer programming needed to be taught before college, to prevent and address negative STEM stereotypes (Markert 1996; Madill et al. 2007). By 2011, the inclusion of computer science education in the K-12 curriculum was seen as key for "succeeding in a technological society, increasing interest in the information technology professions, maintaining and enhancing U.S. economic competitiveness, supporting inquiry in other disciplines, and enabling personal empowerment" (National Research Council 2011).

As computer science education entered federal agencies and the school system, well-funded non-profits such as Code.org championed awareness and access by launching curricular initiatives, educational frameworks, professional development, and policy changes. In 2009, the first Computer Science Education Week was held, which became the Hour of Code in 2013. Today, more than 100 million people have participated in this endeavor (Code.org 2019).

In 2015, the STEM Education Act was passed (H.R.1020 2015) representing the first time that federal funding for STEM was explicitly extended to cover computer science programs. The National Science Foundation launched the STEM + C (Computer Partnerships) program with the goal of "helping all students—but particularly students in science, technology, engineering and mathematics disciplines—need to understand the role of computation and computational thinking within disciplinary problem solving" and "to build the evidence base for effective pedagogy and pedagogical environments that will make the integration of computing within STEM disciplines more age-appropriate and contemporaneously relevant to pre-K-12 STEM education" (de Strulle and Shen n.d.).

The history of the consolidation of STEM as a disciplinary cluster strongly linked to computer science originated in the need to maintain US international primacy by having a strong economy and national security. However, it also restricted the power of computer science education to a limited group of disciplines, to a limited group of students and teachers, and to the particular demands of the workforce.

Pioneers, such as Seymour Papert, the developer of LOGO, and others within the Constructionist movement, could foresee this limitation and claimed that the true power of computer science education was to provide new ways of thinking (Papert 1980; Kafai and Resnick 1996; Resnick 2017; Bers 2018a, b; Resnick et al. 2009). The field of computer science education borrowed pedagogical approaches from STEM disciplines and its instructional methodologies were based on solving challenges designed to learn concepts and skills in a sequenced order of increased complexity. In early childhood, this approach translated into activities such as the ones promoted by the popular Code.org website. These are carefully designed to engage children in solving structured puzzle-like challenges, such as navigating mazes using instructional commands. Lessons in the K-2 sequence feature a series of increasingly more complex mazes that vary in theme, but essentially rely on direction cues to move an object around the screen. Children need to solve the maze and then can move up to the next level.

Although popular, based on the ease of classroom implementation, approaches such as this one reduce the potential of learning how to code to a problem-solving activity, ignoring the expressiveness and communicative functions of programming. Coding is about making meaning by creating a personally meaningful project that can be shared with others. Puzzle-type approaches miss the opportunity to explore the richness of a programming language as a symbol system with a grammar and syntax that can be used to express thoughts and ideas. Furthermore, a few decades ago, Papert suggested that the process of learning to program may be akin to learning a new language (Papert 1987). However, empirical research did not explore this path, neither instructional programs nor curriculum were developed based on this observation. The CAL approach, described next, addresses this.

An alternative perspective: coding as another language

The CAL approach, presented in this paper, is in sharp contrast with the STEM tradition described earlier. CAL's influence can be traced back to two lines of work: First, Constructionism, developed by Seymour Papert (Papert 1980; Bers 2008; Resnick 2017; Kafai and Resnick 1996), which shows that, when children have opportunities to learn a programming language to create computational projects to express themselves, they are likely to encounter powerful ideas from different disciplines, and to think about their own thinking. Second, the long-standing work on literacy instruction, based on research on how young children learn to read and write and the cognitive changes associated with that progression, which provides tools for adapting the processes of learning how to read and write a natural language to an artificial language.

CAL's premise is that coding is a literacy for the 21st century and, as such, it can borrow strategies for teaching other literacies (Bers 2019). Alphabetical literacy enables people to represent and interpret ideas through texts that can travel away from immediate contexts and still be understood by people (Vee 2013). Similarly, algorithms allow people to represent ideas through computer programs that are interpreted by a computer or a robot. Both activities, coding and reading and writing, involve a problem-solving dimension as well as the use and manipulation of a language, a symbolic representational system, to create a sharable, interpretable product. For alphabetical literacy, a natural language. For coding literacy, an artificial language.

Research has explored the similarities and differences between natural and artificial languages and interdisciplinary endeavors such as natural language processing and computational linguistics have emerged (Allamanis et al. 2018). While that work is beyond the scope of this paper, it is important to establish that both natural and artificial languages meet three common criteria: they are meaningful, productive, and allow displacement (Norman 2017). That is to say, languages are symbolic representational systems, with a grammar and syntax, that can be used to convey meaning, to produce something that has never happened before, and to communicate about things that are displaced in time or space. Thus, the end goal of the activity of coding and decoding is to ultimately comprehend, generate, communicate, and express ideas or thoughts by making a sharable product that others can interpret (Bers 2018a). Within this perspective, CAL puts problem solving at the service of personal expression.

CAL's approach and curriculum explores the parallels between programming and natural languages and their communicative and expressive functions. Research shows that children learn to think with and through language (Vygotsky 1978). Thus, by learning to use a programming language that involves logical sequencing, abstraction, and problem solving, children can learn how to think in analytical ways. Wittgenstein (1997) argued that the language we speak determines the thoughts we are able to have. In other words, learning a new language can make new patterns of thought, new conceptual frameworks, and new ways of using language (Wittgenstein 1997). Wittgenstein's philosophy echoes Vygotsky's developmental perspective in terms of the relationship between language and thinking at the individual level. Furthermore, researchers such as Walter Ong, while studying societies that are transitioning from orality to literacy, also found a fundamental shift in their form of thought (Ong 1982).

Just like literacy, coding can change not only the way we think, but also the way we see ourselves in society. Mitchel Resnick and David Siegel, when discussing the creation of the Scratch Foundation to promote an expressive approach to coding, wrote (Resnick and Siegel 2015): "For us, coding is not a set of technical skills but a new type of literacy and personal expression, valuable for everyone, much like learning to write. We see coding as a new way for people to organize, express, and share their ideas … In many introductory coding activities, students are asked to program the movements of a virtual character navigating through a set of obstacles toward a goal. This approach can help students learn some basic coding concepts, but it doesn't allow them to express themselves creatively—or develop a long-term

engagement with coding. It's like offering a writing class that teaches only grammar and punctuation without providing students a chance to write their own stories" (Resnick and Siegel 2015, pp. 1–3).

Papert reminded us that, using this approach, children are not only likely to learn how to program but also to encounter knowledge from other disciplines. For example, in his own work with LOGO, Papert described the encounter with mathematics (Papert 1980). The CAL curriculum puts computer science in direct conversation with powerful ideas from literacy identified by Chall (1983) and other literacy scholars (e.g., Shanahan et al. 2010; Duke and Pearson 2002; Carnine et al. 2006). The process of learning how to program expressively takes time and requires instruction. While children can discover things on their own, a curriculum shows a pathway to expose them to a comprehensive scope and sequence. However, curriculum must be grounded on maturational stages or learning trajectories, developmental models of how children think and operate within a domain (Clements 2007; Clements and Sarama 2004).

I coined the term "coding stages" to describe the learning trajectories in the domain of computer science that young children go through, with increasingly nuanced levels of sophistication, when engaged with an intentional curriculum, such as CAL curriculum. The next section introduces six coding stages and draws parallels to stages of literacy development: emergent, coding and decoding, fluency, new knowledge, multiple perspectives, and purposefulness.

Coding stages

Describing the activity of coding as a learning progression assumes a developmental approach supported by instruction that takes into account both cognitive as well as socioemotional factors. Efforts to describe learning as a progression of stages are influenced by Piaget's work (1952); however, the coding stages presented in this paper depart in an important way. This is not a universal attempt at explaining a naturally occurring phenomenon, as was Piaget's cognitive development stages. It is an effort to create a blueprint to describe a learning path for young children that can be supported through instruction that includes both a curriculum (e.g., CAL), a programming language (e.g., KIBO robotics and/or ScratchJr), and a pedagogical approach [e.g., The Positive Technological Development framework (Bers 2012)].

The instructional elements in the curriculum are designed with a scope and sequence that corresponds to a developmental progression that starts by exploring what is the concept of programming and technology, and culminates with the ability to purposefully create a program to express themselves in a meaningful way. Stages are not fixed and linear, and children can move up and down stages and encounter one or more stages at the same time. The CAL curriculum is explicitly designed to help children move through six coding stages: emergent, coding and decoding, fluency, new knowledge, multiple perspective, and purposefulness. Each and every one of these stages involves the creation of a personally meaningful computational project.

While in early childhood mathematics (Clements and Sarama 2004) and early childhood literacy (Lonigan et al. 2008) there has been extensive research on defining learning progressions and stages, very little work has been done with early childhood computer science. Pilot research explored a developmental model of programming based on observations of children 4–7 years old using KIBO robotics, and categorized children into four stages: proto-programmer, early programmer, programmer, and fluent programmer (Vizner 2017). These exploratory trajectories start with learning that order matters and ends with learning more complex patterns of sequencing such as conditionals and loops. Data from this work show that children's stages did not correspond to age.

This finding is consistent with previous work. While some studies found that children ages 5–6 may have a limited ability to grasp the programming of conditionals (e.g., if–then) (Barrouillet and Lecas 1999; Janveau-Brennan and Markovits 1999), other studies found that age did not correlate with performance on conditional and repeat programs (Elkin et al. 2016; Strawhacker and Bers 2015). While useful, these efforts at creating learning trajectories did not take into consideration the expressive dimension of programming. They mostly focused on the sequencing and problem-solving aspects of programming. The work presented in this paper extends this research by focusing on coding as a literacy; thus, the expressive and communicative dimensions are central in defining coding stages or learning trajectories. Furthermore, research on emergent literacy stages informed the description of the coding stages.

Scholars on emergent literacy have found that children enter school with a great deal of skill and knowledge about reading and writing, although perhaps not in a formal or conventional way (e.g., Ferreiro and Teberosky 1982; Sulzby 1989; Sulzby and Teale 1991; Whitehurst and Lonigan 2001). This early knowledge lays the foundation for later literacy success. This insight is applicable when thinking about coding, even though there is no "orality" period, children are immersed in an interactive technologically rich world, before they are even aware of what programming is and frequently encounter powerful ideas, such as sequencing, cause and effect, correspondence, that are foundational for coding.

Just as children do not begin to talk by speaking in complex utterances, or decode by reading a novel (Chall 1983), children do not begin writing in complete sentences but start by scribbling (Puranik and Lonigan 2011; Ferreiro and Teberosky 1982). Reading and writing are intimately related. Although research on writing has been scarce compared to that on reading, literacy researchers have identified a learning progression or stages that can happen through instruction. The same applies to coding. Children do not start by programming complex algorithms and using nested control structures. They begin with simple sequencing (Lockwood and Mooney 2018; Jenkins 2002; Guzdial and Morrison 2016) and a well-developed curriculum can help them move through more sophisticated stages.

While there is a rich tradition of cognitive scientists, experimental psychologists, and psycholinguists doing basic research on how the brain learns to read (Wolf and Stoodley 2007; Dehaene 2010) and write (Puranik and Lonigan 2011; Bialystok 1991) and there are well-known controversies and theoretical battles based on that research, such as the Reading Wars (Pearson, 2004) and the

Linearity and Unified Hypothesis in writing (Tolchinsky 2003; Fox and Saracho 1990), there is a lack of research on the cognitive mechanisms involved when young children learn to code (Fedorenko et al. 2019). Some research explores the differences between expert and novice programmers (Dalbey and Linn 1985) and other employs tools such as fMRI (Siegmund et al. 2014; Floyd et al. 2017) to characterize mechanisms and propose a theoretical foundation to ground this novel work. However, there is not enough empirical work to be able to categorically identify different stages in the coding learning progression.

The six coding stages presented here are based on behavioral observations and data collection from over two decades of work conducted with young children (4 to 7) learning to code in different settings using a variety of integrated CAL-based curriculum. CAL positions the process of coding as a semiotic act, a meaning making activity, and not only a problem-solving challenge, even during its earliest, most basic levels of instruction. Thus, throughout all six coding stages, the instruction involves children using the programming language to create and share a personally meaningful project. This approach is informed by Constructionism (Papert 1980). This research was done with block-based programming languages, ScratchJr and KIBO, which engage children in sequential thinking and present interfaces that are developmentally appropriate and explicitly designed to promote literacy and expressiveness (Bers 2018b).

The following table (see Table 1) describes the six coding stages and draws parallels with stages of literacy development. However, it is important to note that Chall's stages (Chall 1983) begin with babies and extend beyond college, tapping into the life span. In contrast, the characterization of coding stages proposed here focuses only on early childhood, spanning the 4 to 7 years old range. Although the progression from one coding stage to the next is independent of age, the developmental level of the child informs how quickly the child can progress through the stages.

The CAL approach assumes that teaching to code involves the use of a developmentally appropriate programming language such as KIBO or ScratchJr, designed to both introduce concepts such as loops and conditionals, and also support personal expression and creativity. Given that these are introductory programming languages, it is possible to reach the more complex stages (multiple perspectives and purposefulness) with sufficient instruction and learning time, as well as to begin exploring powerful ideas from the discipline of computer science such as algorithms, modularization, representation, control structures, the design process, debugging, software, and hardware (Bers 2008).

Through the lens of CAL, coding is not only an instrumental tool for problem solving, but a communicative tool for expression. Thus, the pedagogy and the curriculum must scaffold opportunities for the creation of meaningful projects. The following stories present children learning with either KIBO or ScratchJr and exhibiting behaviors that characterize each of the six different stages of coding. These vignettes occurred in the context of a variety of learning settings that utilized different versions of integrated CAL curricula with diverse disciplinary content.

Table 1 The six different coding stages that young children move through when learning to code with CAL, and their corresponding literacy stages

Stages	Coding	Literacy
1. Emergent	Learn the variety of purposes of technology (hardware/software). Recognize that technologies are human engineered Explore what a programming language is and when it is used Learn concepts of interface (e.g., turning on and off, different elements of interfaces, screens, inputs and outputs) Understand the concept of symbolization and representation (i.e., a command is not the behavior, but represents the behavior) Explore basic control structures, such as cause and effect Identify when technologies do not work and the need to problem solve	Learn the variety of purposes of language Explore the structure of the language and recognize some signs Learn concepts of print (e.g., recognize in a story book where the words are on the page, know how to hold a book and a pen) Understand the concept of symbolization and representation (i.e., a noun is not the object itself, but represents an object) Recognize the difference between writing drawing and scribbling
2. Coding and decoding	Learn a limited set of symbols (syntax) and grammar rules within a programming language Understand that sequencing matters and that the order in which commands (symbols) are put together generates different behaviors Create simple programs with simple cause and effect commands Learn to engage in simple debugging by trial and error Identify and fix grammatical errors in the code (i.e., make it work) Instruction is specific to the developmentally appropriate programming language of choice (e.g., KIBO or ScratchJr)	Learn the arbitrary set of letters and begin associating them with corresponding sounds and parts of spoken words and between printed and spoken words Learn the 'alphabetic principle' and the spelling (orthographic) system Acquire general understanding of the spelling-sound system Read and write simple text with high-frequency words and phonically regular words Learn to edit the writing and fix errors Instruction is specific to the particular natural language used
3. Fluency	Consolidation of all that was previously learned Master the full syntax of the programming language and be able to create complex programs using control structures Apply knowledge gained to create complex programs that are personally meaningful Ability to apply all steps of the design process Distinguish and fix logical errors in the code (i.e., a program runs, but it doesn't do what is expected) Transition from 'learning to code' to 'coding to learn'	Accomplish automatic word-recognition and increasingly fluent reading and writing. Consolidation of all that was previously learned. Apply knowledge gained to read and write increasingly complex words and stories. Transition from 'learning to read' to 'reading to learn'' Distinguish between grammatical and semantical errors in a sentence Being able to write complex stories and begin to explore different genres
4. New knowledge	Ability to combine multiple control structures and create nested programs for making a complex project Consolidation of debugging tools, allowing for more strategic debugging Learning how to learn new commands	A qualitative shift in the nature and demands of reading and writing Exposure to new readings and interpretations, and ability to do more complex writing Emphasis is on vocabulary and background knowledge growth

Table 1 (continued)

Stages	Coding	Literacy
5. Multiple perspectives	Understand a situation from the point of view of others and be able to create programs that reflect this understanding Creating programs that involve user's input and that can span different expressive genres (i.e. games, stories)	Reading widely from a broad range of complex materials, both expository and narrative, with a variety of viewpoints. Ability to write from multiple perspectives, utilize different voices, and switch genres easily
6. Purposefulness	Coding is skillfully used for one's own needs and purpose Coding at high levels of abstraction requiring high skill and flexibility Coders analyze, synthesize, and make judgments based on what they read Coding is rapid and efficient	Reading and writing are skillfully used for one's own needs and purpose Reading and writing at high levels of abstraction requiring high skill and flexibility. Readers and writers analyze, synthesize, and make judgments based on what they read and what they want to write Reading and writing is rapid and efficient

Emergent stage

Jenny is 4 years old, and at preschool she receives an iPad with ScratchJr. Jenny is very comfortable with her mom's cell phone, and although she has seen tablets before, she was never allowed to use them on her own. Jenny's teacher told her class that for the next 3 weeks they were going to be using tablets with ScratchJr, a programming language to make animations. Jenny is not exactly sure what a programming language is, but she loves animations. She watches them on TV, as well as on her mom's phone. The invitation of making her own animation is intriguing. During the first lesson, Jenny is told how to turn on and off the tablet, how to position it correctly and how to launch ScratchJr by clicking on the icon that shows a picture of a kitten. She is also told to ask for help if text that she can't read appears on the screen. Jenny's teacher shows the class how to take care of the tablet, which is expensive, and how to put it away in its charging station, so the next class can also use it.

Once finished with the basics, Jenny's teacher connects her own tablet to the projector and shows children an animation she made with ScratchJr. There is an elephant walking in the jungle. The elephant stops when it gets to a puddle and then makes a silly noise (see Fig. 1). Jenny loves it. She finds it very funny. The teacher shows children the different elements she used for making the animation: the backgrounds, the characters, the programming blocks, the paint editor, etc. She then shows them how to add another elephant to her animation and how to program it. She drags two programming blocks with different colors: the green flag, so the

Fig. 1 ScratchJr project of an elephant stopping when it gets to a puddle and then making a silly noise. (Color figure online)

program starts, and a blue "move forward" block. Finally, she presses the green flag on the interface, and the animation starts.

She invites a few children to come to the front and add their own blue motion blocks to her sequence to see what happens. Children find this very intriguing and program the elephants to jump, turn right and left. Finally, the time has come for children to create their own projects with ScratchJr. She explains that first, they will create a simple animation with only two blocks. The class becomes chaotic, but the teacher has anticipated that this would happen. Some children are not able to find the ScratchJr app on their tablet, and others need help to drag the blocks. The few who can follow the teacher's instructions are asked to walk around the room to help those in need. After 10 min, the teacher invites children to save their projects, regardless of their state, and return their iPads. She promises them that during the next few classes they will continue their explorations with ScratchJr.

This vignette shows how Jenny was offered the opportunity to encounter powerful ideas of computer science at the simplest level. She explored the tablet, as hardware, and the ScratchJr app, the software. As the curriculum progressed, Jenny participated in different activities to develop an understanding of the complex interactions between a hardware/software system. This exploration would prove helpful for Jenny to understand her interactions with most of the technologies that surround her in the world. Jenny was also exposed to the concept of representation when the teacher showed her the different elements of the Scratchjr interface and played a game for children to identify the different icons and programming blocks and their meanings. As children created the short program for the elephant, they encountered the foundational concept of algorithms: order matters. The sequence in which we put the blocks dictates the behaviors of the character on the screen. Although this vignette describes a first introductory class to ScratchJr, it is possible to see how Jenny's teacher gave children the opportunity to also engage in the design process. Although this was a beginners' class, children were invited to create a very simple project. They did not know the syntax or the grammar of ScratchJr, but by setting an environment in which those children who were more advanced could play a helper role, Jenny's teacher was able to establish in the classroom an early culture of debugging and problem solving.

In the emergent literacy period, children are exposed to language by seeing written words and participating in word games. They are being read to and asked to look at books and "pretend-read" to become familiar with the "book interface," even though they cannot read on their own yet. Similarly, in the emergent coding stage, children are exposed to the programming language. They are shown programming commands as well as programs written by someone else, and they are given the possibility to learn how to use the interface of the tool to create very simple things. They are immersed in a culture that can program, a culture that can problem solve and debug, in the same way that in the emergent literacy stage children are exposed to a classroom culture that "can" read.

Coding and decoding stage

Liana is 5 years old, and she is sitting with an iPad in her kindergarten class. She is focused. Every so often, she wiggles. She suddenly shouts, to no one in particular, "Look at my cat! Look at my cat!" Liana is excited to show her animation. She has programmed the ScratchJr kitten to appear and disappear. She has put together a long sequence of purple programming blocks. Liana cannot read yet, but she knows that these programming blocks can make her ScratchJr kitten show and hide.

When Liana's kindergarten teacher hears her excitement, she walks over to see Liana's project. Liana is proud to show "my movie," as she calls it. "I made it. Look at my cat. It appears and disappears, it appears and disappears, it appears and disappears. Many times. Look!" She clicks on the green flag on ScratchJr and the animation starts. At that point, Liana's teacher asks her, "How many times does the kitten show and hide?" "Ten times," replies Liana. "I ran out of room. I wanted more times." The teacher shows her an orange programming block, called "Repeat." This block allows for other blocks to be inserted inside its "loop." It then runs the blocks inside the loop as many times as the programmer decides.

After some trial and error, in which Liana plays with inserting different combinations, she figures it out. She chooses the number 99 and clicks the green flag to see the animation. The kitten starts appearing and disappearing. After a few seconds, Liana gets bored of watching. She goes back to her code and changes the number of repetitions to 20.

During this experience, Liana engaged with some of the most powerful ideas of computer sciences that are accessible for a young child. She learned that a programing language has a syntax in which symbols represent actions. She understood that her choices had an impact on what was happening on the screen. She was able to create a sequence of programming blocks to represent a complex behavior (e.g., appearing and disappearing). She used logic in a systematic way to correctly order the blocks in a sequence. She practiced and applied the concept of patterns, which she had learned earlier during math class, which is a precursor for modularity. She discovered the concept of loops and parameters. At the same time, she engaged in problem solving and debugging, and also exercised her tenacity at tackling something she cared about (i.e., having a long kitten movie).

Liana was able to create a project from her own original idea and turn it into a final product. She was personally attached to her movie and proud to share it. Although Liana is in the coding and decoding stage and still learning the syntax of ScratchJr, developing computational thinking for her involved more than problem solving; it meant gaining the concepts, skills, and habits of mind to express herself through "her movie." Within a CAL approach, the curriculum is set up to promote these kinds of opportunities for children to learn the language of programming while expressing themselves and creating personally meaningful project. Similarly, when children are learning to read and write and discover the alphabet and its possible combinations, they are given the opportunity to read "interesting" books and start working on their own books (for example, during writer's workshop) to share with others.

Fluency stage

Maya and Natan are in kindergarten and they are working on a joint KIBO robotics project: to program KIBO to dance the Hokey Pokey. KIBO doesn't use a screen; instead it is programmed by putting together a sequence of wooden blocks with pegs and holes, each representing a command for the robot. Maya starts with the green "begin" block and concludes with the red "end" block. But she has no blocks in between. She can't choose the actions for KIBO because she forgot the KIBO Hokey Pokey song the teacher taught them. Natan, her teammate, reminds her of the song:

You put your robot in
You put your robot out
You put your robot in
And you shake it all about
You do the Hokey Pokey
and you turn yourself around
That's what it's all about!

Maya sings along and, as the song progresses, she chooses the blocks and starts putting them together in a sequence. Begin, "You put your robot in;" forward, "You put your robot out;" backward, "You put your robot in;" forward, "And you shake it all about," shake. She suddenly stops and says, "Natan, I can't find the 'do the hokey pokey' block!" "There is no block for that, silly," responds Natan. "We need to make it up. Let's have KIBO turn on the blue and red light instead. That will be our Hokey

Fig. 2 KIBO and the program to dance the Hockey Pockey. (Color figure online)

Pokey block." Maya agrees, adds those two blocks to the sequence and also adds "shake," "spin," and "beep" to represent the "what it's all about" part of the song. Maya and Natan look at their program while singing the song to make sure they have all the needed blocks. Then they turn on KIBO to test things out. The red light of KIBO's scanner (the "mouth," as Maya calls it) is flashing, meaning that the robot is ready to scan each of the barcodes printed on the wooden programming blocks (see Fig. 2).

Natan takes his turn and scans the blocks one by one. He goes too fast and skips the "red light" block. Maya points that out and he restarts scanning. The children are excited to see their robot dance the Hokey Pokey. "When I count to three, you start singing," says Maya to Natan. They know the drill. They have practiced it during technology circle time in class. Natan sings and both KIBO and Maya dance the Hokey Pokey. KIBO dances too fast. "Can you sing faster?" asks Maya. Natan tries one more time, but it still doesn't work. "We have a problem." he says. "I can't sing fast enough to keep up with KIBO." Maya has an idea. For each action in the song she puts two blocks, so KIBO's motions will last longer. For example, for the line "you put your robot in," she uses two "forward" blocks instead of just one, and so on for each of the commands. Natan tries singing again and this time KIBO dances at the right pace.

Both children start clapping, shaking their bodies and jumping up and down. This vignette, described previously in other work (Bers 2018a) shows how, without knowing it, children who were in the fluent coding stage could engage with many powerful ideas of computer science, such as sequencing, algorithmic thinking, and problem solving. They could focus their attention on the big picture, not just on individual commands, and use their imagination and fluency to decide how they wanted their robot to dance the Hokey Pokey. In addition, because they were able to integrate their coding knowledge into a project that required some calculation, children also explored math concepts they were learning in kindergarten, such as estimation, prediction, and counting. In literacy terms, this is equivalent to going from "learning to read," to "reading to learn." That means, they were "learning to code" and were able to transition to "coding to learn"—in this case, learning mathematical ideas of pacing and duration. Their fluency level was expressed by the unique way they created their Hokey Pokey dance.

New knowledge stage

Madison is a 7-year-old girl who has been programming with ScratchJr for 1 year both at school and at home. As Madison begins a new project, she plans it aloud, adding new characters and actions as she needs them. She declares she is going to program a basketball game with multiple players. As she explores the ScratchJr library for backgrounds, she narrates her design process: "I'm going to have many teammates, and a crowd cheering, and a dragon. And the game will happen in a gym." Madison chooses the gym background and opens the paint editor. She draws a rectangle in the corner of the gym, and paints it brown. That will be her snack stand.

Next, Madison adds ten players to her basketball game. Some she chooses from the character library; some she draws herself with the paint editor. There are girls and boys and animals and fantasy characters. As she formulates her design she expresses an understanding of the tool: "I want the kitten to pass the ball to the girl. So, I'll have to program the ball to move forward, when I tap on the girl." Clearly, Madison understands that ScratchJr is not magic—the characters have potential to do many things, but will only *actually* do what she *programs* them to do. There are multiple ways for her to program this, but she chooses to program the girl to send a blue message to the ball when someone taps on her. When the ball receives the blue message, it will move forward. Madison is in control. She understands she must choose the programming blocks in the correct sequence to get the characters to carry out the desired actions. She also knows there is not only one way to do the programming. She can choose the way she likes best.

Later, Madison wants the dragon, which she has colored purple, to dribble the ball, jump, and shoot. This requires knowledge of sequencing, cause and effect, and an awareness of the debugging process if the characters do something different than she intended. Madison programs the dragon to move right five times to the basketball hoop, then hop. The basketball's program proves trickier. At first, she programs it to move right then hop in a repeat loop. This results in a rigid motion that does not look like dribbling. "No! I want it to move forward and bounce at the same time!" Madison exclaims. After engaging in a trial and error process involving several combinations, she has a breakthrough: "I can make two programs at once! Cool!" She writes two separate programs for the ball, one that tells it to move right, and one that tells it to hop. They both start when she taps the green flag. Madison discovered the fundamental computer science concept of parallelism.

As Madison keeps working on her project, she draws and programs a crowd of animal fans to cheer by recording three sound blocks. Now, her project has images and sounds, as well as movement. Madison explored sequencing, debugging, modularization, and the design process, some of the core ideas of computational thinking. But she also gained new knowledge, such a discovering the possibility of having two programs working in parallel. Madison solved problems to make her basketball game and learned new knowledge, but what kept her engaged was her desire to tell a story about her favorite sport: basketball. The new knowledge stage is characterized by the ability of children to learn new concepts and skills because they are committed to make a project they truly care about. This has similarities with literacy. Children will read books with increasing difficult words and sentence constructions, because they are passionate about the stories they tell. It is this emotional connection with the material that engages children in new learning.

Multiple perspectives stage

Alma and Ben are 7 years old and they are participating in a ScratchJr summer camp. During the first few days of camp, they have learned to program interactive stories, but today, the project is different. The camp counselor, Matt, invites them to make a ScratchJr game for everyone to play with. There are twelve kids in their

group, so it is hard to imagine all of them with a single iPad. Alma and Ben are confused. Matt reminds them of the "Lights Around the World" project they saw a few days ago to celebrate the Chinese New Year. The dragon and the firecrackers moved across multiple iPad screens that were put next to each other on a long table. Matt told them that they could use multiple tablets for their game, but they had to make a game for everyone to play—not an animation for people to watch.

Alma and Ben brainstorm for a long time how they could create a game with multiple tablets. Suddenly, Ben has an idea. He remembers a memory matching game with colorful cards he has at home. Each card displays a different fruit on one side, and the name of the game on the other side. All the cards are first set upside down. They all look alike, displaying the name of the game. Players need to guess which cards, when turned up, will be matched and display the same two fruits. If they guess correctly, they keep those cards. However, if the cards do not match, they need to be put back on the table upside down for the next player to guess. The winner is the player who can take the most pairs of matching fruits. Players who remember the location of the different fruits are more likely to win because they know how to choose the right cards more efficiently.

Ben describes the game to Alma and together they discuss how they can program a ScratchJr memory matching game. Alma proposes to make a game with animals, not fruits. They take multiple tablets with ScratchJr and use the paint editor to draw an animal in each of them. They keep track of their drawings to make sure they have an even number of tablets, with two animals that are the same on pairs of tablets. Alma spends a long time trying to replicate the beautiful lion she drew on a second tablet. Ben shows her how she can use the same lion for a second tablet using the "airdrop" function. Once Alma discovers this, she is fast at making all of the other animals. She now has to draw each of them only once. That saves her time and effort.

After half an hour, Ben and Alma have twelve tablets with six animals. They call Matt, who asks them: "What do you think the player can do with these iPads with animals? What will be fun?" Ben proposes that the tablets would be turned upside down, so the animals are not visible, and the players would turn them up and discover the matching animals. Alma doesn't like the idea. "That is not a new game. That is exactly like the game you have but with iPads instead of cards." Alma doesn't articulate it clearly, but she realizes that there is no programming involved in this game. That is what makes ScratchJr unique.

After a long debate, in which Matt, the counselor, intervenes by asking questions, the children morph their memory game into a Whack-A-Mole game. That is, they program each animal to make a noise once it is tapped on. If the animals tapped are the same, the noise will also be the same. Both children work very hard at making the game and arrange all twelve tablets on the floor. They invite their classmates to come and play with the newly created "Whack-An-Animal-Noise" game. On Alma's mark, every child presses the green flag on the tablet in front of her, and observes the screen displaying different animals one by one. When two animals are displayed at once in two different tablets, they have to quickly reach across the table and tap those animals. If they are successful, they win. It takes another day for the game to

work well, but children in the summer camp do not mind trying different versions and suggesting improvement.

Alma and Ben designed a game by drawing on their experience with other games. They had advanced ScratchJr knowledge and they were able to put that knowledge to use to create something new. Furthermore, in order to make a fun, interactive game, they had to use the ScratchJr programming blocks to invite user's interaction. In this case, the "Start-on-Tap" block proved extremely useful. The process of understanding how to design a project for someone else to interact with involves not only sophisticated programming skills, but also the ability to engage in perspective taking. This is similar to the decentering process and the ability of writers and readers to switch voices in their texts.

Purposefulness stage

This vignette shows how, once children become experts with programming and have mastered all the stages of coding: emergent, coding and decoding, fluent, new knowledge and multiple perspectives, they can create projects to meet set purposes and goals, while also expressing themselves. Mark and Sarah are in first grade. In social studies, they have been learning about the Iditarod dog sled race, held annually in Alaska to reenact the 1925 transportation of a medical serum across the state to combat a large-scale diphtheria epidemic.

On their classroom wall, there is a huge map of Alaska marked with the different checkpoints across the state, from Willow to Nome. Mark and Sarah learned about geography by studying the Iditarod race and its different routes. They also learned that back in 1925, a safe route was organized, and the 20-pound cylinder of serum was sent first by train, and then relayed by twenty mushers and more than 100 dogs that ran in relays.

Mark and Sarah studied Alaska's towns and geography, as well as the history of the epidemic and the designated safe routes. They have been doing research on the subject for over 2 weeks. But today it is their time to put all of that knowledge to use: not by passing an exam or completing a worksheet, but by re-creating the Iditarod race with KIBO robots.

Mrs. Dolan gives them the challenge to build and program their robots to travel from one checkpoint to another, starting in Willow and ending in Nome, carrying all the things mushers must carry, as well as the "pretend" serum for sick children. Each team receives a piece of thick cardboard with two checkpoints marked at the ends and a KIBO robot. They first need to draw the route from checkpoint to checkpoint and decorate the cardboard with the geography of that region. Second, they need to build their robots with a platform that can carry everything needed, including a safe way to transport the serum until reaching the next checkpoint and passing it on to the next team.

Mrs. Dolan puts the cardboard pieces together on the floor in the school library, making a huge floor map of Alaska. Mark starts to decorate the cardboards with snow, trees, mountains, and a family of foxes. Sarah builds a robot with two motors

and wheels at the sides and a moving platform on top. She adds a light bulb, and two sensors: an "ear" to detect sound and an "eye" to detect light. The robot is ready to go, but it must be programmed, otherwise it will not move. Mark wants the robot to follow the path he drew on the cardboard. He is hoping the eye sensor will be able to pick up the dark marker trace. They try a few times, but it doesn't work. The children decide to try the sound sensor. They program KIBO to go forward and turn right every time it detects a sound. Children start clapping to direct the KIBO robot but quickly realize that sometimes they need to turn left, not right, and they have programmed it to always turn right when there is a clap.

A few exchanges follow, in which children are busy trying out different strategies. After some trial and error, they make it work. Now they are ready to make the journey a little bit fancier. They decide that the robot will turn its blue lightbulb before moving, signaling that the serum is on board. They also decide that before arriving to the last checkpoint it will shake and turn its red light on to alert the next team to get ready.

Mark and Sarah are expert KIBO programmers and were able to experiment with different sophisticated approaches for their robot to travel following a path. While children with less knowledge would have used the "counting method," counting how many forwards blocks are needed from one check point to the next, and the "backwards methods" drawing the path for the robot on the board, after figuring out the program, Mark and Sarah choose to use sensors. They were already fluent at programming with them, so they did not have to worry about the technical aspects of how to do it, and they could focus on best approaches. Furthermore, they were not only able to solve the problem ahead of them—to safely carry the serum from one checkpoint to the next—but they were also able to express their creativity by incorporating the use of the lightbulb.

Just like with literacy, when a child reaches the purposefulness stage, she has the intellectual tools to decide how and when to apply the learned skills to fulfill not only someone else's purpose (e.g., the author's goal when writing the text; or the teacher's challenge), but their own purpose (e.g., to interpret the text and to add personalize the challenge).

In summary, a child's pathway from the emerging to the purposefulness coding stage is not linear. Some children might go back and forth between stages as they learn new concepts and skills, and some might be fluent with certain powerful ideas and programming concepts, but not others. But throughout all stages, the CAL curriculum invites the child to use her developing coding knowledge to create an expressive project and to share it with others.

The CAL curriculum

The CAL curriculum is designed for children 4 to 7 years old to be used in both formal and informal learning settings. It supports the transition through the six coding stages described earlier, by exposing children to developmentally appropriate powerful ideas of computer science as well as to principles of literacy. Guided by the Positive Technological Development framework, the curriculum targets the whole

child, by proposing activities that involve cognitive as well as socioemotional and moral engagement.

From a pedagogical perspective, CAL is informed by three different approaches. First is Constructionism (Papert 1980), which conceives of computer programming as an opportunity for children to learn new things by making personally meaningful projects. Second is Positive Technological Development (Bers 2012), which proposes that learning experiences using computer programming must engage children in six positive behaviors (six C's): content creation, creativity, communication, collaboration, community building, and choices of conduct. Third is Dialogic instruction (Clarke et al. 2015; Alexander 2008; Littleton and Howe 2010; Resnick et al. 2010), which proposes that instruction happens best when there are opportunities for students to engage in authentic explanation and argumentation and open-ended interpretation about the subject matter, in this case computer science.

The CAL curriculum is organized into four units, all centered around a children's book, and are designed to engage emergent readers or early readers in expressive programming using the KIBO robotics kit or the tablet-based ScratchJr. The curriculum units, regardless of the technology used, follow similar structure and include time spent working with coding and literacy as well as an emphasis on off-screen activities involving social interactions, creativity, and movement. Individual and group activities in this curriculum include warm up games to playfully introduce or reinforce concepts, design challenges to solidify skills, free explorations to allow students to tinker and expand their skills, expressive explorations to promote creativity, writing activities and technology circles to share and reflect on activities. The culmination of each unit is an open-ended project to share with family and friends.

Each unit contains twelve 1-h lessons that allow children to explore storybooks such as Where the Wild Things Are by Maurice Sendak and There Was an Old Lady Who Swallowed a Fly by Simms Taback. For example, children might program a robot to do a wild rumpus dance, recalling special moments from the books they have read, or they might write and animate their own alternative story endings in ScratchJr. The lessons identify powerful ideas from both computer science and literacy and are aligned to academic frameworks of Common Core literacy standards (National Governors Association for Best Practices 2010), as well as K-12 CS frameworks (K-12 Computer Science Framework Steering Committee 2016).

The term powerful idea refers to a central concept or skills within a discipline that is simultaneously personally useful, inherently interconnected with other disciplines, and has roots in intuitive knowledge that a child has internalized over a long period of time (Papert 1980). The powerful ideas from computer science addressed in this curriculum include algorithms, design process, representation, debugging, control structures, modularity, and hardware/software. See Table 2 for a comparison of each curriculum in terms of the teaching of coding.

The powerful ideas from literacy that are placed in conversation with these powerful ideas from computer science are the writing process, recalling, summarizing and sequencing, using illustrative and descriptive language, recognizing literary devices such as repetition and foreshadowing, and using reading strategies such as predicting, summarizing, and evaluating. Teachers are encouraged to use the CAL curriculum as a guiding resource and to adapt lessons and activities to their needs

Table 2 Emergent and Reader's curriculum units for both KIBO and ScratchJr

Tools	Emergent readers	Programming blocks	Readers	Programming blocks
KIBO	Book: *There Was an Old Lady Who Swallowed a Fly* by Simms Taback		Book: *Where the Wild Things Are* by Maurice Sendak	
Lesson 1:	Foundations	Unplugged	Foundations	Unplugged
Lesson 2:	Technologies and Robots	Begin, End, Blue Motion	Technological Tools	Begin, End, Blue Motion
Lesson 3:	Sequencing	Beep, Sing + *Begin, End, Blue Motion*	Sequencing	Beep, Sing + *Begin, End, Blue Motion*
Lesson 4:	Taking Care of Our Materials	Yellow Light + *Begin, End, Blue Motion, Beep, Sing*	Programming	Yellow Light + *Begin, End, Blue Motion, Beep, Sing*
Lesson 5:	Programmer and Author	*Begin, End, Blue Motion, Beep, Sing, Yellow Light*	Debugging	*Begin, End, Blue Motion, Beep, Sing, Yellow Light*
Lesson 6:	Programming	*Begin, End, Blue Motion, Beep, Sing, Yellow Light*	Cause and Effect—Level 1	Wait for Clap, Sound Sensor, Orange Sound Recorder + *Begin, End, Blue Motion, Beep, Sing, Yellow Light*
Lesson 7:	Debugging	*Begin, End, Blue Motion, Beep, Sing, Yellow Light*	Cause and Effect—Level 2	*Begin, End, Blue Motion, Beep, Sing, Yellow Light, Wait for Clap, Sound Sensor, Orange Sound Recorder*
Lesson 8:	Cause and Effect	Wait for Clap, Sound Sensor + *Begin, End, Blue Motion, Beep, Sing, Yellow Light*	Repeat Loops—Level 1	Repeat, End Repeat, Number parameters + *Begin, End, Blue Motion, Beep, Sing, Yellow Light, Wait for Clap, Sound Sensor, Orange Sound Recorder*
Lesson 9:	Repeat Loops	Repeat, End Repeat, Number parameters + *Begin, End, Blue Motion, Beep, Sing, Yellow Light, Wait for Clap, Sound Sensor*	Repeat Loops—Level 2	Light and Distance sensor, gray sensor parameter + *Begin, End, Blue Motion, Beep, Sing, Yellow Light, Wait for Clap, Sound Sensor, Orange Sound Recorder, Repeat, End Repeat, Number parameters*

Table 2 (continued)

Tools	Emergent readers	Programming blocks	Readers	Programming blocks
Lesson 10:	Final Project—Characterization	*Begin, End, Blue Motion, Beep, Sing, Yellow Light, Wait for Clap, Sound Sensor, Repeat, End Repeat, Number parameters*	If Statements	If, End If, purple sensor parameters + *Begin, End, Blue Motion, Beep, Sing, Yellow Light, Wait for Clap, Sound Sensor, Orange Sound Recorder, Repeat, End Repeat, Number parameters, Gray sensor parameter*
Lesson 11:	Final Project—Retelling	*Begin, End, Blue Motion, Beep, Sing, Yellow Light, Wait for Clap, Sound Sensor, Repeat, End Repeat, Number parameters*	Final Project—Writing the Wild Rumpus Composition	*Begin, End, Blue Motion, Beep, Sing, Yellow Light, Wait for Clap, Sound Sensor, Orange Sound Recorder, Repeat, End Repeat, Number parameters, gray sensor parameter, If, End If, purple sensor parameters*
Lesson 12:	Final Project—Expansion	*Begin, End, Blue Motion, Beep, Sing, Yellow Light, Wait for Clap, Sound Sensor, Repeat, End Repeat, Number parameters*	Final Project—Coding the Wild Rumpus	*Begin, End, Blue Motion, Beep, Sing, Yellow Light, Wait for Clap, Sound Sensor, Orange Sound Recorder, Repeat, End Repeat, Number parameters, gray sensor parameter, If, End If, purple sensor parameters*
ScratchJr	Book: *Knuffle Bunny* by Mo Willems		Book: *Giraffes Can't Dance* by Giles Andreae and Guy Parker-Rees	
Lesson 1:	Taking Care of Materials	Unplugged	Foundations	Lesson 1: Unplugged
Lesson 2:	What Is a Program?	Begin, End, Blue Motion	What Is a Program?	Begin, End, Blue Motion + Go to Page End
Lesson 3:	Beginning, Middle & End	Begin, End, Blue Motion	Sequencing	*Begin, End, Blue Motion*
Lesson 4:	Programmers and Authors	*Begin, End, Blue Motion*	Characters	Purple Blocks + *Begin, End, Blue Motion*
Lesson 5:	What Is a Character?	*Begin, End, Blue Motion*	Programming	*Begin, End, Blue Motion, Purple Blocks*

Table 2 (continued)

Tools	Emergent readers	Programming blocks	Readers	Programming blocks
Lesson 6:	Characters Can Talk!	Record + *Begin, End, Blue Motion*	Debugging	*Begin, End, Blue Motion, Purple Blocks*
Lesson 7:	What Is the Setting?	Go To Page End + *Begin, End, Blue Motion, Record*	Details	Speed, Wait Time + Begin, End, Blue Motion, Purple Blocks
Lesson 8:	Sequencing	*Begin, End, Blue Motion, Record, Go to Page End*	Cause and Effect—Part 1	Repeat + Begin, End, Blue Motion, Speed, Wait Time, Purple Blocks
Lesson 9:	High Five Retell	*Begin, End, Blue Motion, Record*	Cause and Effect—Part 2	Start on Tap, Sound Recorder + *Begin, End, Blue Motion, Speed, Wait Time, Purple Blocks, Repeat*
Lesson 10:	Self Portraits—I Can be a Character!	*Begin, End, Blue Motion, Record*	Cause and Effect—Part 3	Message, Start on Message, Start on Bump + *Begin, End, Blue Motion, Speed, Wait Time, Purple Blocks, Repeat, Start on Tap, Sound Recorder*
Lesson 11:	Story Time—Part 1	*Begin, End, Blue Motion, Record*	Final Project—Part 1	*Begin, End, Blue Motion, Speed, Wait Time, Purple Blocks, Repeat, Start on Tap, Sound Recorder, Message, Start on Message, Start on Bump*
Lesson 12:	Story Time—Part 2	*Begin, End, Blue Motion, Record*	Final Project—Part 2	*Begin, End, Blue Motion, Speed, Wait Time, Purple Blocks, Repeat, Start on Tap, Sound Recorder, Message, Start on Message, Start on Bump*

Programming blocks that hadn't previously been introduced, are not italicized and are at the beginning of the programming blocks list

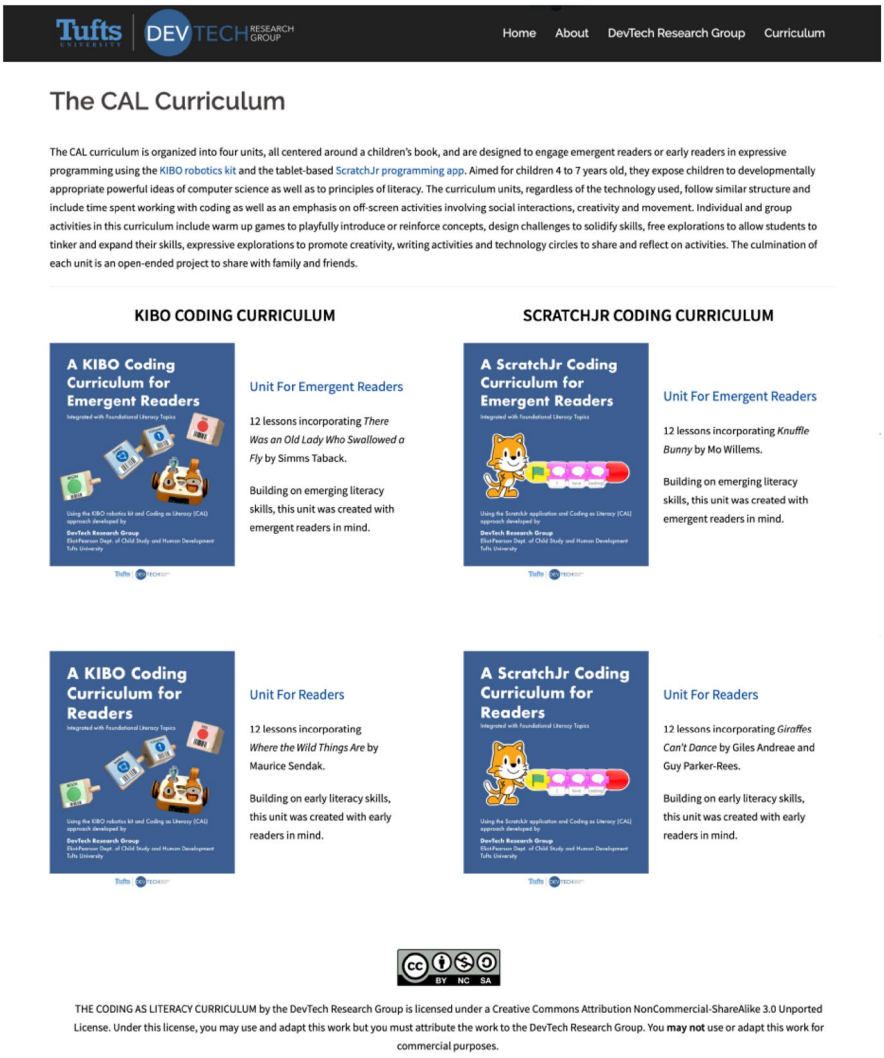

Fig. 3 Screen capture of the website hosting the CAL Curriculum

of their students, as well as choose their own favorite books. The CAL curriculum can also be downloaded as PDF documents and is explicitly designed to help children move through six coding stages: emergent, coding and decoding, fluency, new knowledge, multiple perspective, and purposefulness.

The free curriculum can be accessed through a website (URL: https://sites.tufts.edu/codingasliteracy/), see Fig. 3, that catalogues the units based on reading level and the programming language that is being used (the KIBO robotics kit or the tablet-based ScratchJr). In addition to a summary of each lesson in the units, the website also includes videos and tutorials to assist teachers by providing model lessons, as they

navigate through the curriculum, curricular resources, and teaching materials such templates for design journals, as well as assessment tools.

Conclusion

Computer science education is growing and expanding to the early years. However, it is not enough to copy models used in later schooling—which mostly grew out of the STEM disciplines. Programming languages and pedagogies need to be developmentally appropriate for young children. Language plays an important role in early childhood, a time in which children are learning to read and write. The CAL approach described here leverages the teaching of literacy by broadening the range of languages children are exposed to, including programming languages. Just like with natural languages, learning how to program involves learning how to use a symbolic system, its syntax and grammar, to express and communicate ideas.

The field of literacy education has developed instructional strategies based on research-based evidence that shows learning trajectories in the development of reading and writing to become a literate person. The field of early computer science education is just starting to emerge and therefore coding stages are not yet clearly defined or thoroughly investigated. It might be, for example, that the six coding stages presented here for early childhood could also apply to programming novices of any age.

The goal of this paper is to present a different approach for computer science education in early childhood, CAL, which supports the teaching of programming as a literacy by providing a scope and sequence of curricular activities that help children move through six different coding stages or learning progressions. The vignettes in this paper are not intended to fully characterize each of these stages, but to demonstrate the basic principles that root the CAL approach, likening programming fluency to literacy development. In summary, CAL is based on the following principles:

a. Strategies used in literacy education can be helpful for teaching children how to code.
b. Coding projects can provide opportunities for children's sense-making and expression.
c. Problem solving can serve as a means toward self-expression and communication.
d. Coding activities can engage children in thinking about powerful ideas from computer science, as well as other domains.

A conceptualization of programming that is not solely STEM-based may help combat the stigma associated with STEM disciplines and attract a wider range of children to computer science. Thus, decades of scholarly work and teaching practices on language development and reading and writing instruction can provide new pathways that inform the early teaching of computer science.

If education aims at helping people think creatively to solve the problems of our world, only a subset of those problems can be solved by STEM disciplines. As more people learn to code and computer programming leaves the exclusive domain of computer science to become integral to other professions, it is more important than ever that we develop computer science pedagogies that promote deep and thorough engagement for everyone starting in early childhood.

Acknowledgements The author is deeply thankful to members of the DevTech research group at Tufts University, and to Ziva Hassenfeld for discussions of these materials, Amanda Strawhacker and Anne Drescher for help with manuscript editing, and Riva Dhamala for help with table and formatting.

References

Abelson, H., & DiSessa, A. (1981). *Turtle geometry: The computer as a medium for exploring mathematics (The mit press series in artificial intelligence)*. Cambridge, MA: MIT Press.

Alexander, R. J. (2008). *Towards dialogic teaching: Rethinking classroom talk*. Cambridge: Dialogos.

Allamanis, M., Barr, E. T., Devanbu, P., & Sutton, C. (2018). A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR), 51*(4), 81.

Barrouillet, P., & Lecas, J. (1999). Mental models in conditional reasoning and working memory. *Thinking & Reasoning, 5*(4), 289–302.

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

Bers, M. (2012). *Designing Digital experiences for positive youth development: from playpen to playground*. Oxford: Oxford University Press.

Bers, M. (2018a). *Coding as a playground: Computational thinking and programming in early childhood*. London, UK: Routledge.

Bers, M. U. (2018b). Coding, playgrounds and literacy in early childhood education: The development of KIBO Robotics and ScratchJr. In *2018 IEEE Global Engineering Education Conference (EDUCON), 2100*.

Bers, M. (2019). Coding as another language: Why computer science in early childhood should not be stem. In *Key Issues in Technology and Early Childhood* (Editor Chip Donohue). NY: Routledge.

Bers, M., & Resnick, M. (2015). *The official ScratchJr Book: Help your kids learn to code*. San Francisco, CA: No Starch Press.

Bialystok, E. (1991). Letters, sounds, and symbols: Changes in children's understands of written language. *Applied Psycholinguistics., 12,* 75–89.

Bredekamp, S. (1987). *Developmentally appropriate practice in early childhood pro- grams serving children from birth through age 8*. Washington, DC: National Association for the Education of Young Children.

Carnine, D. W., Silbert, J., Kame'enui, E., Tarver, S., & Jungjohann, K. (2006). *Teaching struggling and at-risk readers: A direct instruction approach*. Upper Saddle River, NJ: Pearson.

Chall, J. S. (1983). *Stages of reading development*. New York: McGraw-Hill.

Clarke, S., Resnick, L. B., & Rosé, C. P. (2015). *Dialogic instruction: A new frontier* (3rd ed., pp. 378–389). New York: Handbook of Educational Psychology.

Clements, D. H. (2007). Curriculum research: Toward a framework for research-basedCu rricula. *Journal for Research in Mathematics Education, 38*(1), 35–70.

Clements, D. H., & Sarama, J. (2004). Learning trajectories in mathematics education. *Mathematical Thinking and Learning, 6,* 81–89.

Code.org. (2018). 2018 annual report. Seattle, WA. Retrieved from https://code.org/files/annual-report-2018.pdf.

Code.org. (2019). https://code.org/.

Cunha, F., & Heckman, J. (2007). The technology of skill formation. *American Economic Review, 97*(2), 31–47.

Dalbey, J., & Linn, M. C. (1985). The demands and requirements of computer programming: A literature review. *Journal of Educational Computing Research, 1*(3), 253–274.

Dehaene, S. (2010). *Reading in the Brain: The new science of how we read*. New York: Penguin Books.
de Strulle, A., & Shen, C. (n.d.). STEM + Computing K-12 Education (STEM + C). *National Science Foundation*. Retrieved from https://wwwnsf.gov/funding/pgm_summ.jsp?pims_id=505006.
Duke, N., & Pearson, P. D. (2002). Effective practices for developing reading comprehension. In A. Farstrup & S. Samuels (Eds.), *What research has to say about readinginstruction* (3rd ed., pp. 205–242). Newark, DE: International Reading Association.
Elkin, M., Sullivan, A., & Bers, M. U. (2016). Programming with the KIBO robotics kit in preschool classrooms. *Computers in the Schools, 33*(3), 169–186. https://doi.org/10.1080/07380569.2016.1216251.
Fayer, S., Lacey, A., & Watson, A. (2017). *BLS spotlight on statistics: STEM occupations-past, present, and future*. Washington, DC: U.S. Department of Labor, Bureau of Labor Statistics.
Fedorenko, E., Ivanova, A., Dhamala, R., & Bers, M. U. (In press). The language of programming: A cognitive perspective. *Trends in Cognitive Development.*
Ferreiro, E., & Teberosky, A. (1982). *Literacy before schooling*. Exeter, NH: Heinemann.
Floyd, B., Santander, T., & Weimer, W. (2017). Decoding the representation of code in the brain: An fMRI study of code review and expertise. In *Proceedings of the 39th International Conference on Software Engineering* (pp. 175–186). IEEE Press.
Fox, B., & Saracho, O. (1990). Emergent writing: Young children solving the written language puzzle. *Early Child Development and Care., 56,* 81–90.
Gadanidis, G. (2017). Five affordances of computational thinking to support elementary mathematics education. *Journal of Computers in Mathematics and Science Teaching, 36*(2), 143–151.
Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher, 42*(1), 38–43. https://doi.org/10.3102/0013189X12463051.
Guzdial, M. (2008). Education: Paving the way for computational thinking. *Communications of the ACM, 51*(8), 25–27. https://doi.org/10.1145/1378704.1378713.
Guzdial, M., & Morrison, B. (2016). Seeking to making computing education as available as mathematics or science education. *Communications of the ACM, 59*(11), 31–33.
Heckman, J., & Masterov, D. (2007). The productivity argument for investing in young children. *Review of Agricultural Economics, 29*(3), 446–493.
Hubwieser, P., Armoni, M., Giannakos, M. N., & Mittermeir, R. T. (2014). Perspectives and visions of computer science education in primary and secondary (K-12) schools. *ACM Transactions on Computing Education, 14*(2), 7.
Janveau-Brennan, G., & Markovits, H. (1999). The development of reasoning with causal conditionals. *Developmental Psychology, 35*(4), 904–911.
Jenkins, T. (2002). On the difficulty of learning to program. In *Proceedings of the 3rd Annual. Conference of the LTSN Centre for Information and Computer Sciences* (pp. 53–58). Leeds, UK. Retrieved from http://www.psy.gla.ac.uk/~steve/localed/jenkins.html.
K-12 Computer Science Framework Steering Committee. (2016). K–12 computer science framework. Retrieved from https://k12cs.org.
Kafai, Y. B., & Resnick, M. (1996). *Constructionism in practice: Designing, thinking, and learning in a digital world*. Mahwah, NJ: Erlbaum.
Littleton, K., & Howe, C. (2010). *Educational dialogues: understanding and promoting productive interaction*. London: Routledge.
Lockwood, J., & Mooney, A. (2018). Computational thinking in education: Where does it fit? A systematic literary review. *International Journal of Computer Science Education in Schools, 2*(1), 41–60. https://doi.org/10.21585/ijcses.v2i1.26.
Lonigan, C. J., Schatschneider, C., & Westberg, L. (2008). Developing early literacy: Report of the National Early Literacy Panel. Washington, DC: *National Institute for Literacy. Identification of children's skills and abilities linked to later outcomes in reading, writing, and spelling* (pp. 55–106).
Madill, H., Campbell, R. G., Cullen, D. M., Armour, M. A., Einsiedel, A. A., Ciccocioppo, A. L., et al. (2007). Developing career commitment in STEM-related fields: Myth versus reality. In R. J. Burke, M. C. Mattis, & E. Elgar (Eds.), *Women and minorities in science, technology, engineering and mathematics: Upping the numbers* (pp. 210–244). Northhampton, MA: Edward Elgar Publishing.
Markert, L. R. (1996). Gender related to success in science and technology. *The Journal of Technology Studies, 22*(2), 21–29.
National Governors Association Center for Best Practices, Council of Chief State School Officers. (2010). *Common Core State Standards.* National Governors Association Center for Best Practices, Council of Chief State School Officers, Washington, DC.

National Research Council. (2011). *Report of a workshop of pedagogical aspects of computational thinking*. Washington, DC: National Academy Press.
National Research Council. (2012). A framework for K-12 science education: Practices, crosscutting concepts, and core ideas. Committee on a Conceptual framework for new K-12 science education standards. Board on Science Education, Division of Behavioral and Social Sciences and Education. Washington, DC: The National Academies Press.
National Research Council Committee on Early Childhood Pedagogy, Bowman, B., Donovan, S., & Burns, M. (2001). *Eager to learn: Educating our preschoolers*. Washington, DC: National Academy Press.
Norman, K. L. (2017). *Cyberpsychology: An introduction to human-computer interaction*. Cambridge: Cambridge University Press.
Ong, W. J. (1982). *Orality and literacy: The technologizing of the word*. London: Methuen.
Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books Inc.
Papert S. (1987). Computer criticism vs. technocentric thinking *Educational Researcher* (Vol. 16, No. I) January/February 1987.
Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology, 2,* 137–168.
Pearson, P. D. (2004). The reading wars. *Educational policy, 18*(1), 216–252.
Pei, C., Weintrop, D., & Wilensky, U. (2018). Cultivating computational thinking practices and mathematical habits of mind in lattice land. *Mathematical Thinking and Learning, 20*(1), 75–89. https://doi.org/10.1080/10986065.2018.1403543.
Perlis, A. J. (1962). The computer in the university. In M. Greenberger (Ed.), *Computers and the world of the future* (pp. 180–219). Cambridge, MA: MIT Press.
Piaget, J. (1952). *The origins of intelligence in children* (Vol. 8, p. 18). New York: International Universities Press.
Puranik, C., & Lonigan, C. (2011). From scribbles to scrabble: Preschool children's developing knowledge of written language. *Reading and Writing, 24*(5), 567–589.
Resnick, M. (2017). *Lifelong kindergarten: Cultivating creativity through projects, passion, peers, and play*. Cambridge: MIT Press.
Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., et al. (2009). Scratch: programming for all. *Communications of the ACM, 52*(11), 60–67.
Resnick, L. B., Michaels, S., & O'Connor, C. (2010). How (well-structured) talk builds the mind. In D. Preiss & R. Sternberg (Eds.), *Innovations in educational psychology: Perspectives on learning, teaching and human development* (pp. 163–194). New York: Springer.
Resnick, M., & Siegel, D. (2015). A different approach to coding. *International Journal of People-Oriented Programming, 4*(1), 1–4.
Ryan, M. (2011). *The encyclopedia of literary and cultural theory*. Hoboken, NJ: Wiley-Blackwell.
Shanahan, T., Callison, K., Carriere, C., Duke, N., Pearson, D., Schatschneider, C., & Torgesen, J. (2010). Improving reading comprehension in kindergarten through 3rd grade: ies practice guide. NCEE 2010-4038. *What Works Clearinghouse*.
Shonkoff, J., Phillips, D., & National Research Council (U.S.). Committee on Integrating the Science of Early Childhood Development. (2000). *From neurons to neighborhoods: The science of early child development*. Washington, DC: National Academy Press.
Siegmund, J., Kästner, C., Apel, S., Parnin, C., Bethmann, A., Leich, T., & Brechmann, A. (2014). Understanding source code with functional magnetic resonance imaging. In *Proceedings of the 36th International Conference on Software Engineering* (pp. 378–389). ACM.
STEM Education Act of 2015, House of Representatives 1020, 114th Congress. (2015). Retrieved from https://www.congress.gov/bill/114th-congress/house-bill/1020.
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, 25*(3), 293–319.
Sulzby, E. (1989). Assessment of writing and of children's language while writing. In L. Morrow & J. Smith (Eds.), *The role of assessment and measurement in early literacy instruction* (pp. 83–109). Prentice-Hal: Englewood Cliffs, NJ.
Sulzby, E., & Teale, W. (1991). Emergent literacy. In R. Barr, M. Kamil, P. Mosenthal, & P. D. Pearson (Eds.), *Handbook of reading research* (Vol. 2, pp. 727–758). New York: Longman.
Tolchinsky, L. (2003). *The cradle of culture and what children know about writing and numbers before being taught*. Mahwah, NJ: Lawrence Erlbaum Associates.

Vee, A. (2013). Understanding computer programming as a literacy. *Literacy in Composition Studies, 1*(2), 42–64. https://doi.org/10.21623/1.1.2.4.

Vizner, M. Z. (2017). *Big robots for little kids: Investigating the role of sale in early childhood robotics kits* (Master's thesis). Available from ProQuest Dissertations and Theses database. (UMI No. 10622097).

Vygotsky, L. S. (1978). *Mind in society: The Development of higher psychological processes*. Cambridge, MA: Harvard University Press.

Whitehurst, G., & Lonigan, C. (2001). Emergent literacy: Development from prereaders to readers. In S. B. Neuman & D. K. Dickensen (Eds.), *Handbook of early literacy research* (pp. 11–29). New York: Guilford Press.

Wilson, C., Sudol, L. A., Stephenson, C., & Stehlik, M. (2010). *Running on empty: The failure to teach K-12 computer science in the digital age*. New York, NY: The Association for Computing Machinery and the Computer Science Teachers Association.

Wing, J. (2006a). Computational thinking. *Communications of Advancing Computing Machinery, 49*(3), 33–36. https://doi.org/10.1145/1118178.1118215.

Wing, J. M. (2006b). Computational thinking. *Communications of the ACM, 49*(3), 33–35.

Wing, J. (2011). Research notebook: Computational thinking—What and why? The link magazine, Spring. Carnegie Mellon University, Pittsburgh. Retrieved from https://www.cs.cmu.edu/link/research-notebookcomputational-thinking-what-and-why.

Wittgenstein, Ludwig. (1997). *Philosophical Investigations. Trans. G.E.M. Anscombe* (2nd ed.). Cambridge: Blackwell. Print.

Wolf, M., & Stoodley, C. J. (2007). *Proust and the squid: The story and science of the reading brain* (1st ed.). New York, NY: HarperCollins.

Marina Umaschi Bers is a professor and chair at the Eliot-Pearson Department of Child Study and Human Development and an adjunct professor in the Computer Science Department at Tufts University, where she heads the interdisciplinary Developmental Technologies (DevTech) research group. She also developed and serves as director of the graduate certificate program on Early Childhood Technology at Tufts University. Her research involves the design and study of innovative learning technologies to promote children's positive development, most specifically in early childhood. She co-designed the ScratchJr programming language and she developed the KIBO robot kit for children 4 to 7 years old, which can be programmed with wooden blocks without using keyboards or screens. She received a MEd from Boston University and an MS and PhD from the MIT Media Laboratory working with Seymour Papert. Her philosophy, pedagogical and theoretical approach can be found in her latest book "Coding as Playground: Programming and Computational Thinking in the Early Childhood Classroom" (Routledge, 2018).