SPECIAL ISSUE

Computer Science K–8: Building a Strong Foundation

csta Computer Science Teachers Association

SPECIAL ISSUE

Computer Science K–8: Building a Strong Foundation

Computer Science Teachers Association

SPECIAL ISSUE
CS K–8

Computer Science Teachers Association
Association for Computing Machinery
2 Penn Plaza, Suite 701
New York, New York 10121-0071

For additional copies please contact: *cstapubs@csta.acm.org*

csta Computer Science Teachers Association

FORWARD

Over the past few decades, computers have transformed both the world our students live in and the world of work and innovation that they will join in the future. No area of human endeavor or enterprise will be untouched by technology...technology that will require technological tool builders, as well as, tool users. More than ever, to be well educated and to lead empowered lives our students will require a sound foundation in the principles and practices of computer science. There is no time too soon to begin the process of building that foundation.

Recently, CSTA has witnessed a growing interest in computer science at the elementary and middle school levels. In some ways, increased recognition of the importance and relevance of computer science stems from a growing awareness of the need for computational thinking skills for all students and the importance of embedding this learning throughout the school experience. Excellent new teaching tools and curricula are also making computer science learning more accessible and engaging for students of all ages.

This publication draws together articles previously published in the CSTA Voice, as well as newly-commissioned pieces by key CSTA thought leaders. Our hope is that this collection will inspire innovative thinking, offer new ideas, and provide practical strategies for ensuring that computer science concepts are deeply woven into all students' learning experiences throughout their formal and informal education.

On behalf of CSTA, we wish to thank all of the authors who have so kindly contributed to this resource and all of our members who continue to strive to prepare their students to thrive and excel in the new global economy.

Pat Phillips, *Editor*

Steve Cooper, *Chair, CSTA Board of Directors*

Chris Stephenson, *Executive Director*

csta Computer Science Teachers Association

CSTA MISSION

The Computer Science Teachers Association is a membership organization that supports and promotes the teaching of computer science and other computing disciplines. CSTA provides opportunities for K–12 teachers and students to better understand the computing disciplines and to more successfully prepare themselves to teach and learn.

SPECIAL ISSUE
CS K–8

csta Computer Science Teachers Association

SPECIAL ISSUE CS K–8

Making the Case for CS Education in K–8

Patrice Gans

Over 25 years ago, Seymour Papert published his landmark book, *Mindstorms: Children, Computers, and Powerful Ideas*. In the forward, "The Gears of My Childhood," Papert first proposed the idea that computers would revolutionize education. "The computer is the Proteus of machines. Its essence is its universality, its power to simulate. Because it can take on a thousand forms and can serve a thousand functions, it can appeal to a thousand tastes" (Papert, 1980).

The power of the computer is its ability to be something for everyone. Computing is not a goal in and of itself, but the means to an end, enabling students to take control, solve problems, and build a future based upon their imagination and creativity. Given the opportunity to provide students with this power and open doors to untold opportunities, what teacher wouldn't jump at the chance?

Seymour Papert, a mathematician and protégé of Jean Piaget (early 20th Century developmental psychologist, who observed that children acquire knowledge by acting on the world around them), created a practical and powerful way for teachers to implement constructivist learning in the elementary classroom. In 1967, Papert designed LOGO, the first programming language specifically for children. Through LOGO, Seymour Papert demonstrated that children can learn best when they use computers in a way that puts them in the active roles of designers and builders. With these and other computer science (CS) tools, teachers now have the implements to help young children develop powerful competency in the four C's (Critical thinking and problem solving, Communication, Collaboration, and Creativity and innovation) and begin to develop the "algorithmic" or "computational thinking" skills needed to meet 21st Century challenges.

Skills integral to CS learning are valued and measured across the curriculum. Since the 1960s, CS has been shown to be an ideal addition to the elementary school curriculum. All of the four C's skills develop when students are engaged in CS projects and activities. Students must internalize the content of a subject before they can "compute" the knowledge into a computer game, multimedia production, or computer program. As demonstrated in the articles in this special CSTA compendium, examples abound. Multimedia stories can be crafted in language arts classes, multi-level computer games can be designed in mathematics, and art and music projects can be delivered through electronic media.

Rarely are such activities solo events; communication and collaboration are integral components of any K–8 computing project. The give-and-take common in collaborative work around a complex project involves more than just one-way communication of ideas; it requires focused listening, reforming ideas, and more.

And problem-solving takes center stage. Students discover that the first step in problem-solving is to state the problem clearly and unambiguously. Students also find they must learn to navigate the iterative design process inherent in a computing activity.

1. Start with that clearly-stated challenge.
2. Gather information.
3. Formulate a plan.
4. Create a working prototype.
5. Experiment and debug
6. Gather feedback from others.
7. Revise and redesign
8. Publish and begin again.

Traditionally, students are first exposed to CS in high school Advanced Placement (AP) CS if they are fortunate enough to be at a school that offers a rigorous CS course. Unfortunately, female and minority students do not enroll in numbers representative of their proportion of the population. According to Computing in the Core coalition (of which CSTA is a founding member) only 17% of AP CS test-takers in 2008 were women, although women represented 55% of all AP test-takers and 51% of the population. In addition, participation in AP CS tests among underrepresented minorities has increased in the past 10 years, but is only 11%, compared to 19% of all AP test-takers. K–8 CS education may be able to help reverse these trends by illustrating for all students the potential for personal power afforded by computing skills.

As an elementary school technology teacher, I know from experience that students can develop a love for computing, sophisticated skills, and a desire to learn more during their primary school years. Researchers are considering the possibility that early exposure will translate into increased enrollment and a life-time engagement in CS. It is believed that positive experiences at an early age will motivate students to further explore the opportunities that a CS education has to offer. It's important to take advantage of this developmental stage to build a CS literate and functional population. It is in everyone's best interest and we have the tools to make it happen.

This CSTA publication, *Computer Science K–8: Building a Strong Foundation*, provides a review of current thought on K–8 CS education, explores how CS topics and concepts can impact learning in the K–8 classroom, and offers practical strategies and resources. Here you will find updates on research from higher education, including a study at Harvey Mudd College on a new approach to teaching middle school CS curriculum, ideas on teaching CS skills by playing criminal detective using databases, suggestions for planning afterschool and summer CS camps, plans for integrating CS into classes as diverse as fine arts and mathematics, and personal experiences of dozens of classroom teachers using a wide variety of tools and techniques.

We hope that these articles will engage and inspire you so that you might do the same for your students.

LEARN MORE:

Papert, S. (1980) *Mindstorms: Children, computers, and powerful ideas*. New York, NY: Basic Books.
CSTA K–12 Computer Science Standards *csta.acm.org/Curriculum/sub/K12Standards.html*
ACM K–12 CS Model Curriculum, 2nd Edition *csta.acm.org/Curriculum/sub/CurrResources.html*

A Philosophy for Children and Computing

Mark Dorling, Daniel Mace, and Roger Sutcliffe

Traditional philosophy dealt with fundamental questions of life such as "What should I do?" (ethics) and "How should we decide?" (politics), as well as, "What can I be certain of?" (knowledge). In the 20th Century these same questions were narrowed down and applied to specific areas of life—resulting in studies such as Philosophy of Religion, Philosophy of Language, and others. There is now a developing branch of philosophy dealing with computer science, with courses due to open at a number of United Kingdom (UK) universities in the autumn of 2012.

While these new branches of philosophy, focusing on specific content, have been opening up in the past 40 years or so, there has been another development in philosophy, extending the range of people who engage with it. Specifically, we are talking about children; their practice of philosophy or philosophizing is usually referred to as Philosophy for Children, or P4C, for short. This is a simple but powerful way of developing children's reflective and critical-thinking skills (and philosophies!) for learning and for life.

In a changing technological society it is vital that we prepare children to not only use technology but to be reflective about how it works, and its use by themselves and others. Mark Dorling, project coordinator of Langley Grammar School's Digital Schoolhouse project, and Roger Sutcliffe, one of the world's leading experts in P4C and past President of SAPERE (Society for Advancing Philosophical Enquiry and Reflection in Education), have developed an exciting variation on this theme: Philosophy for Computing.

These Philosophy for Computing lessons have been designed and delivered by Daniel Mace, teacher of Advanced Skills at Langley Grammar School. By collaborating with Mark and Roger, Daniel has created a series of structured, yet fun and challenging, sessions based on the accelerated learning model. These lessons incorporate critical thinking skills, high-order creativity, and co-construction of ideas to address philosophical issues relating to:

- addiction to computers and the Internet,
- artificial intelligence,
- digital finger print and hacking on the Internet,
- CIS data and privacy,
- advertising on the Internet,
- censorship on the Internet,
- eSafety and Internet grooming/preying, and
- security.

For example, the first session focused on the digital fingerprint that is left on the Internet by users as they click on various websites and give personal information. The first half of the session focused on the everyday psychology behind how advertisements work, before showing how the Internet magnified the

CSTA RESOURCES OF SPECIAL INTEREST FOR K–8

CSTA K–12 Computer Science Standards A core set of standards for a CS curriculum and its implementation.

Curriculum Resources Materials and tools for meeting the learning standards described in the CSTA K-12 Computer Science Standards.

Brochures, Posters, Videos Resources for promoting CS fields and careers.

***Source* Web Repository** A database of instructional materials that can be searched by grade level, title, author, or subject.

issues involved. Students were shown various video Internet advertisements and asked to arrange them according to their appeal. This generated a short discussion on how advertisements are designed to target personality. Paired-group work focused on the skills needed to create personalized advertisements for their partners using a series of random words to prompt student thinking, and then, designing picture postcards of their advertisements using creative drawing prompts.

After a short break these ideas were applied to the Internet. An information race (a comprehension game based on a short piece of text) helped students learn how the Internet collects information from various sources, such as Facebook, Google searches, and others. Students were then asked to filter out the most important sources and identify the most dangerous. The consequent discussion exposed students' use of websites, their thoughts about the dangers, and the beginnings of a class code that encouraged their peers to use the Internet more safely. Finally, a clip from *Minority Report*, a film that showcases a world in which advertisements continuously target people's minds as they walk down the street, brings home this idea to students and provides a stimulated P4C debate about their own roles in feeding the potential power of the Internet.

The end product of these lessons is that pupils develop their own codes of practice for safer and more responsible use of every-day and emerging technologies. This is achieved through nurturing a trust between pupils and teachers to create an environment where pupils are more willing to talk freely about their own experiences and share their concerns with the class. This allows the students to establish their own shared understanding and priorities, which hopefully will create a long-lasting effect.

The techniques described in this article have been applied to students in both Primary (7–10 years of age) and Secondary (11–16 years of age) with great success! For more information about this work into the teaching of the ethics of computing, please email Mark Dorling at: *dsh@lgs.slough.sch.uk*.

LEARN MORE:

Digital School House ***www.digitalschoolhouse.org.uk***
SAPERE ***sapere.org.uk***

Defining Computational Thinking for K–12

Chris Stephenson and Valerie Barr

When Jeannette Wing launched a discussion regarding the role of computational thinking (CT) across all disciplines, she ignited a profound engagement with the core questions of what computer science is and what it might contribute to solving problems across the spectrum of human inquiry. Wing argued that advances in computing allow researchers across all disciplines to envision new problem-solving strategies and to test new solutions in both the virtual and real world.

In the summer of 2009, the Computer Science Teachers Association (CSTA) and the International Society for Technology in Education (ISTE) began a multi-phase project supported by the National Science Foundation, aimed at developing an operational definition of CT for K–12.

Developing an operational definition of, or approach to, CT that is suitable for K–12 is especially challenging because there is, as yet, no widely-agreed-upon definition of CT. In addition, to be useful, this definition must ultimately be coupled with examples that demonstrate how CT can be incorporated in the classroom. The primary work of the project was therefore carried out during two workshops; the first focused on developing a shared understanding of CT, and the second on creating exemplar resources and strategies that would support the implementation of CT concepts and skills across grade levels and subject areas.

In attempting to define what distinguishes CT from other problem-solving methods, the educators involved in this project (more than 30 educators representing all grade levels and multiple subject areas) focused on the centrality of the computer and a set of concepts encompassed by CT.

> *"CT is an approach to solving problems in a way that can be implemented with a computer. Students become not merely tool users but tool builders. They use a set of concepts, such as abstraction, recursion, and iteration, to process and analyze data, and to create real and virtual artifacts. CT is a problem-solving methodology that can be automated and transferred and applied across subjects."*

They also envisioned CT manifesting in the classroom through active problem-solving. They saw students "engaged in using tools to solve problems", "comfortable with trial and error", and working in "an atmosphere of figuring things out together."

The project has already resulted in the creation of a number of useful artifacts, perhaps the most important being this operational definition of CT.

CT is a problem-solving process that includes (but is not limited to) the following characteristics:

- *formulating problems in a way that enables us to use a computer and other tools to help solve them;*
- *logically organizing and analyzing data;*
- *representing data through abstractions such as models and simulations;*
- *automating solutions through algorithmic thinking (a series of ordered steps);*
- *identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources; and*
- *generalizing and transferring this problem-solving process to a wide variety of problems.*

It has also led to identification of a number of dispositions or attitudes that are essential dimensions of CT. These dispositions or attitudes include:

- confidence in dealing with complexity,
- persistence in working with difficult problems,
- tolerance for ambiguity,
- the ability to deal with open-ended problems, and
- the ability to communicate and work with others to achieve a common goal or solution.

In addition to these definitions, the project team created a number of resources to support the implementation of CT in K–12. These include:

- *Computational Thinking Teacher Resources* collection that includes a CT vocabulary and progression chart, nine CT Learning Experiences, and CT classroom scenarios;
- *Computational Thinking Leadership Toolkit*, as a companion piece to the *Computational Thinking Teacher Resources*, includes making the Case for Computational Thinking, Resources for Creating Systemic Change, and a guide for implementation strategies;
- a one-page flier describing CT; and
- key articles written by members of the project team and researchers.

All of these resources are available on the CSTA website. We strongly encourage you to review these resources and join the conversation about how we can best enable all of our students to incorporate CT concepts and skills into their knowledge base. This project is supported by the National Science Foundation.

LEARN MORE:

Computational Thinking Resources *csta.acm.org/Curriculum/sub/CompThinking.html*

Middle School CS? Yes!

Zachary Dodds, Mike Erlinger, and Elizabeth Sweedyk

"There's no recession here!" chirps an Apple Store employee, adding me to the bottom of untold screens of names. In light of the diverse group who overflow the cavernous space, it's hard to argue with his contrived cheerfulness. Here, headscarves jostle with ball caps and shirt-sleeves and ink sleeves crisscross the crowd-gathering, flickering displays. Nowhere is the American ideal of equal opportunity more fully realized than in consumption, and computation is at the top of many consumers' lists.

A generation ago, computation was a tool with relatively little impact on day-to-day life. That generational gap remains today—not in the consumption of computation, however, but in its creation. As a result, computational enthusiasm and computational skills are unbalanced. The Department of Labor employment forecasts confirm this imbalance between the computational skills of our schools' graduates relative to the career opportunities available. The imbalance appears, too, in who develops computational skills. The relative absence of women and ethnic minorities in computer science (CS) is well documented in a variety of studies. And in far too many schools, CS does not appear in curricula until late in high school, if at all, and by the time it does, many students have been convinced that it is not something that *people like them* do.

How and when *should* computation appear in school? Certainly, computation *as a tool* should permeate K–12 education; in light of modern life, there's probably no way to stop this from happening, even if we wanted to. Yet for an initial exposure to computation *per se*, middle school offers us an opportunity to make computational creativity as much a part of students' identities as its consumption is already. At Harvey Mudd College (HMC) we have undertaken two efforts to integrate middle school teachers and students into the culture of creating computation. The first approach is through the design and deployment of educational games, an NSF project entitled *The Games Network: Games for Students, Games by Students*. Our second effort is a middle school curriculum named *Middle-years Computer Science*, or *MyCS*.

The Games Network addresses the misunderstandings students may have about the practice of CS. Our approach is to engage middle school students in a semester-long software development project carried out with college-level CS students. The deliverable is one of the modern-day CS artifacts that all students relate to: an educational computer game.

At the start of each semester, middle school teachers provide HMC's software development class with a list of learning objectives; e.g., items drawn from the sixth and seventh Grade-level Content Expectations (GLCE). Each four-person software development team chooses a GLCE they will target with their game. The middle school teachers and students are integrally involved throughout the development, providing bi-weekly written evaluations of the game's concept, storyline, and user interface. Middle school students also act as testers of game prototypes and beta releases. This first-hand involvement dispels CS stereotypes and offers insights into some of computer science's most compelling challenges.

For middle school teachers, participation does not require any specialized background. Yet it allows them to:

- acquire media-rich, interactive learning tools designed specifically for their classrooms, at no cost;
- engage their students in designing tools for their own learning; and
- support literacy efforts by providing an authentic context for communication between their students and college students/faculty.

For CS professors who teach (or want to teach) game design and development, HMC's Games Network model allows them to:

- enhance game projects by providing their students with a real customer who has an authentic need for game software,
- cultivate a sense of social responsibility in their students and allow them to produce software that can have positive impact, and
- provide their students with the opportunity to serve as role models for younger students.

The design and testing of games provides a natural link with students' everyday experience of computation. Our middle school CS curriculum, *MyCS*, pushes this connection further by making computation itself "the game." *MyCS*'s goal is to develop greater computational awareness and sophistication of larger numbers of middle school students than existing curricula do. Though not truly exclusive to middle school, *MyCS* does not shy away from the turmoil of individual- and group-identity formation that energizes middle school students. Rather, *MyCS* seeks to build bridges amidst the family and societal pressures that influence the long process of self-definition. As students construct their many answers to "Who am I?" they necessarily also answer "Who am I not?" For its part, *MyCS* seeks to place CS in the former category more often than the latter.

Resources for engaging students with the "game of computation" abound, but many of those resources lack an overarching structure. For instance, the online games FactoryBalls and LightBot provide compelling, minds-on introductions to procedural thinking—with the same requirement for precision and clarity that general-purpose computer programming demands. *MyCS* supplements these kinds of resources with our own Picobot, a Karel-like automaton that introduces an abstract language—with all of the concomitant benefits and drawbacks—into students' specifications.

MyCS's hands-on activities similarly convey the building blocks of reproducible procedures and abstraction. In one such exploration, student teams design a small LEGO structure, which they then encode with a set of descriptive labels: the coordinates, shape, orientation, and color of each brick. Students proceed to represent those labels with a binary code, again of their own design. With the binary codes and their explanations, the teams cover their designs and swap their descriptions. Each pair then strives to build the structure specified by their classmates' codes. The activity can have telephone-like results as amusing differences arise between original and rebuilt structures. In this case, however, students can explicitly track down the source of any errors (and correct them) whether within the encoding, context building, context interpretation, or decoding phases of the process. These experiences map immediately to executable software, regardless of architecture or language.

To bring curricular structure to these kinds of activities, *MyCS* builds on the broad shoulders of *Exploring Computer Science (ECS)*. UCLA's innovative effort to democratize CS for tenth- through twelfth-grade students, ECS has enjoyed a successful deployment in high schools throughout the greater Los Angeles area. For its audience, *MyCS* shortens the *ECS* curriculum, emphasizing and expanding ECS's hands-on activities that ask students to create and shape computation. A pilot workshop, supported by Google and the National Science Foundation in summer 2011, brought together teachers of many different grades, CS undergraduates, and CS professors in order to create a first-draft *MyCS* curriculum, based on ECS. Three schools piloted the resulting one-semester offering in the fall of 2011, with an expanded effort slated for spring of 2012.

Already, the experiment has demonstrated the many challenges that remain, as well as the rewards for grappling with them. One challenge is political: how should a curriculum like *MyCS* fit into the well-established technology courses already in existence? Our vision is that *MyCS* would supplement, not replace, those courses; *MyCS* can offer insights into CS *as a discipline* to complement skill-development in specific (and rapidly changing) suites of computational tools. Details differ from school to school. In Claremont, for instance, one section of the district-required technology course has dedicated a day each week to *MyCS* materials. These "CS Fridays" have become a course highlight, popular with teachers and students alike.

A second challenge is communication: maintaining a high-bandwidth feedback loop among middle and high school teachers, college CS students, and instructors. A summer workshop is only a starting point. We must cultivate resources such as student-staffed homework hotlines, out-of-the-box student demonstration days, and equipment-lending libraries that support curricular activities like *ECS*'s computer-disassembly "scavenger hunt." Harvey Mudd College's experiments with these collaborations show that, when resources are available, teachers and students at the middle school level express a demand for CS comparable to the recession-proof pull of CS's artifacts.

Like its students, middle school CS is in the early stages of self-definition, weaving among the pros and cons of a bombardment of questions: "Should computational creativity be part of the curriculum?" "Should it be a choice or a requirement?" "How do we balance CS itself with its influential toolkits?" "Does CS detract from—or can it, perhaps, enhance—middle school students' journeys of choosing who they are?" All of these questions remain open, to be sure, but we believe that efforts like *MyCS* and *The Games Network* can help students, teachers, and colleges answer them to the mutual benefit of all of those groups.

LEARN MORE:

Occupational Outlook Handbook ***www.bls.gov/ooh***
Margolis, J. et al. (2008). *Stuck in the shallow end*. MIT Press.
FactoryBalls ***www.bartbonte.com/factoryballs***
LightBot ***armorgames.com/play/2205/light-bot***
Parlante, N. et al. (2010). Proceedings from *SIGCSE 2010: Nifty assignments*. New York, NY.
Exploring Computer Science ***www.exploringcs.org***
MyCS ***www.cs.hmc.edu/~cs5grad/MyCS***

Programming in Kindergarten

A Playground Experience

Marina Umaschi Bers

In collaboration with Mitchel Resnick at the MIT Media Lab, and with funding from the National Science Foundation, the DevTech research group at Tufts University is working on a new version of Scratch specifically designed for children 5–7 years of age. Scratch Jr. is in its first stages of development and pilot testing and will be released to the public within the next year and a half. This effort to bring computer science to very young children is based upon research into developmentally appropriate programming experiences.

Playgrounds are popular amongst five-year olds. They are designed to support the exploration of the physical environment and the development of motor skills, to engage children in creative open-ended play, and to promote social interactions. They are also probably one of the few spaces where children of this age can be autonomous while exposed to minimal risks, such as bumps and scratches.

Now, think of a playpen. Playpens, big or small, serve to corral children into a confined space. Playpens are in sharp contrast with playgrounds. They are risk-free, but there is no room for autonomous exploration. The adult is in control of the toys and the play experience. There is no room for imaginative play.

In contrast with the playground, the playpen serves as a metaphor to convey the lack of freedom to experiment, explore, be creative, and take risks. The playground promotes, while the playpen hinders, a sense of mastery, creativity, self-confidence, problem-solving, and open exploration.

While most of today's software marketed for young children remind us of playpens and not playgrounds, software that engages them in programming, when presented in a developmentally appropriate way, can provide a wonderful playground experience. Most software is marketed as educational because young children can develop pre-academic dispositions and learn about shapes, colors, letters, sounds, and numbers. However, from a developmental perspective, those are not the most important milestones for children in this age range. This is a time for free exploration, testing boundaries, socializing, taking risks in a safe way, engaging in pretend play, and solving problems autonomously.

In the DevTech Research Group that I direct, we are experimenting with developmentally appropriate programming languages for early childhood education that allow similar kinds of experiences that children encounter in the playground. For the last six years we have been working on the CHERP system (Creative Hybrid Environment for Robotic Programming), that allows young children to program with physical interlocking wooden blocks and to transition back and forth between the screen-based and the tangible programming language. This hybrid approach allows them to work with multiple representations, in the same way that they learn math or literacy by using different media.

As important as our design of CHERP, is the theoretical approach that guides our use of it in the early childhood classroom, the Positive Technological Development (PTD) framework. PTD provides an overall playground approach and guides the design not only of a programming language developmentally appropriate for young children, but also a curriculum, classroom assessment tools, and a professional development component for early childhood educators who are not sure how to welcome computer programming in their classrooms, given their mandate to focus on literacy and math.

Over the years, young children have used CHERP to build LEGO towns with robots that stand upright and wave their arms to greet town visitors and ballerinas that can sing and dance. Some have created robotic flowers that grow out of the ground when there is light and plants that spin as people approach. Many children have created sleds to recreate the Iditarod race in Alaska and others have made soccer players that can kick a ball and trains to transport animals to the zoo. They have also used all sorts of recyclable materials to decorate and expand the robotic hardware.

At the DevTech group we take a playground approach to programming. There is playful learning, autonomous decision-making (even if as adults we know that it will sometimes lead to initial failure), and risk-taking. Children engage in social interactions and negotiations. Children work on the floor, on the table, and on the computer and navigate among those physical spaces.

Programming in early childhood can engage children in a playground experience that serves the fundamental developmental needs of young children. And it can also be a gateway to explore the notion of sequencing, which is a fundamental building block for mathematics and literacy. Developmentally appropriate programming software, such as CHERP and Scratch Jr., can be wonderful playgrounds for young children because they encourage problem-solving, logical thinking, creativity, and love of learning through playful explorations.

LEARN MORE:

DevTech Research Group ***ase.tufts.edu/DevTech***

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. *Playpen to playground*. New York, NY: Oxford University Press.

K-8 Standards

Inconsistency from State to State

Editor's note: High school computer science (CS) education standards vary widely from state to state. The picture of K–8 CS standards shows even greater inconsistency. To illustrate the situation, we interviewed three educators from diverse educational environments and locations to learn about the status of K–8 CS standards in their states.

OHIO: Dave Burkhart teaches computer technology at Sheridan High School and is an adjunct faculty member at Zane State College. He currently serves as the Policy Task Force Chair for CSTA.

NORTH CAROLINA: Deborah Seehorn has worked as a business, finance, and information technology education consultant at the North Carolina Department of Public Instruction for thirteen years. She previously taught mathematics, business, and computer programming. Deborah also serves as the State Department Representative on the CSTA Board of Directors.

TEXAS: Karen North served on the Texas Technology Application (TA) Standards CS writing team. She lobbied to have CS concepts included in K–8 technology essential knowledge and skills. The Texas Essential Knowledge and Skills (TEKS) now include programming languages as tools, and designing a computer program as an activity, in the TA standards of writing a sequence of steps. Previously, she taught high school CS, math, and technology systems, and was an elementary school technology specialist.

Does your state have standards for content that could be considered K–8 CS standards (based on the learning outcomes in the *CSTA K–12 Computer Science Standards*)?

OHIO: Ohio has Technology Standards which are similar to CS standards (*www.ode.state.oh.us/GD/Templates/Pages/ODE/ODEDetail.aspx?page=3&TopicRelationID=1707&ContentID=1279&Content=109741*).

NORTH CAROLINA: There are Information and Technology Standards for K–12 (*www.ncpublicschools.org/acre/standards/new-standards/#it*) that cover all K–12 curricula. The focus of the 6–8 Business, Finance, and Information Technology curricula is on computer skills and applications, and exploratory courses (*www.ncpublicschools.org/cte/business/curriculum/programs*). The Computer Skills and Applications course has been recently revised and does not have a blueprint of essential standards, but course topics can be found by following the above link. The course consists of 12 modules, and the local school systems can teach any modules, in any combination. The other middle-level course, Exploring Business Technologies, does have a course blueprint of standards. There is a careers unit and information technology activities.

TEXAS: Texas has added CS standards to the TEKS which mandate a variety of topics be taught within the required technology application curriculum (*www.tea.state.tx.us/index2.aspx?id=8192*). The standards are available at *www.ittybittyurl.com/texasteks*. I wrote a set of expectations intended to provide teachers with ideas for implementing the standards. They are not part of the TEKS, but an additional resource for teachers that incorporate ideas from the *CSTA K–12 Computer Science Standards*. The State Board of Education has added computational thinking concepts to the Texas standards for mathematics education.

How typical is it that the standards are actually addressed in the classroom?

OHIO: Teaching CS standards in K–8 varies from teacher to teacher, building to building, and district to district. There is no real consistency. The technology standards for K–8 are meant to be integrated into the reading, writing, mathematics, science, and social studies curricula.

NORTH CAROLINA: There is a better-than-average chance that the standards are addressed in the classroom. The business curriculum in the middle grades is taught in all middle-level business programs, but the schools have the flexibility to pick and choose which standards best meet the needs of their students.

TEXAS: The CS technology standards will be new to Texas teachers after the adoption process is complete in the fall of 2012. Early feedback indicates that teachers are glad to have programming included in the standards because it could mean getting the resources they need for their classrooms.

How are teachers prepared to teach content related to technology standards?

OHIO: Recent education graduates from Ohio colleges and universities have had more formal training to address the standards than other teachers, for whom the level of training typically depends upon personal interests. Teachers depend upon professional development provided by their districts, the eTech Ohio Conference, and the CSTA Ohio Chapter.

NORTH CAROLINA: Grade 6–8 business education teachers are fairly well prepared through Career and Technical Education (CATE) training at the state level. The Instructional Technology Division and leaders in each local system provide training for the Information and Technology Standards implementation.

TEXAS: Teachers are trained to include the ICT technology standards, but few receive CS training. The Texas Education Agency has contracted with one of the regional service centers to provide professional development. We may, however, require some advocacy effort to ensure that classroom teachers can access relevant professional development.

Does the state make teaching resources or curriculum materials available to teachers?

OHIO: The state provides some examples, but local educational service centers create materials and resources for teachers in their member schools.

NORTH CAROLINA: Resources such as learning activities and content presentations are available to the teachers on a password-protected Moodle PLC site. Some assessment items are also available.

TEXAS: Project Share provides online professional development for TA teachers (*www.epsilen.com/grp/1220137*). CS is considered a TA course and does not receive Karl Perkins funding reserved for CTE courses. Budget cuts have reduced the number of resources available to CS teachers.

What could your state do to better provide quality CS education to K–8 students?

OHIO: More funding and more professional development would improve technology instruction. However, technology is often a lower priority due to the pressures of testing. About 15–20 years ago, Ohio provided computers to some classrooms through the SchoolNet program. Each year, computers were provided to a specific grade level, beginning with kindergarten classrooms; the money ran out in the year in which the seventh- and eighth-grade classrooms would have received computers. Generally, little state money is available. Most of the responsibility for replacing or adding new computers falls to the discretion and financial ability of local districts.

NORTH CAROLINA: In a state with a large rural population, access to technology resources K–8 is an issue. Until the digital divide is closed implementation will continue to be a struggle.

TEXAS: Require university teacher education programs to include computational thinking activities such as computer programming and make CS part of every masters-level instructional technology program. Labor and business have communicated that CS education is needed for developing the labor market and improving the economy. Perhaps the business community will have a positive impact on CS education. Even though Texas is not part of the Common Core Standards plan, it would help greatly if CS could be included in this national standards movement.

Computer Science: Critical K-8 Learning

Irene Lee

As computer science (CS) educators, we are often faced with skepticism about the necessity of CS education for young students. State education departments and local school districts have been reluctant to include CS education in the K–8 curriculum. Others question if CS education is as important as reading, writing, and arithmetic. This article addresses these concerns by describing the learning objectives of the CSTA K–6 CS standards, why they are critically important for students, and how teachers can implement them in engaging ways.

CSTA's goal in creating the new *CSTA K–12 Computer Science Standards* (2011) was to describe for teachers, administrators, and policy makers the CS knowledge and skills that students must have to enable them to thrive in the 21st Century global information economy. The standards were developed by a team of K–12 educators, university and college faculty, research specialists, and curriculum specialists. Together, they took into account the scope and sequencing of curricula that has been shown to be effective in CS teaching and the developmental trajectories of young learners, as well as the serious constraints many school districts face in terms of teacher training, curriculum innovation, teaching resources, and dissemination. Revisions to the standards were informed by feedback from many sources, including national education and professional organizations, CS educators, and computer scientists. The resulting CSTA standards document provides a framework for CS education at the primary and secondary school levels.

By implementing these standards, schools can introduce the principles and methodologies of CS to all students at all stages of their learning, whether they are college bound or workplace bound. We envision that teachers of K–8 students will use the standards to implement CS activities in the context of other subjects or within stand-alone introductory CS courses. We hope that administrators and policy makers will use the standards to understand the importance of CS education as part of the intellectual development of all students and that all who read the standards will see the important linkages between CS and innovation across disciplines.

The standards contain K–8 learning objectives in five domains (Computational Thinking; Collaboration; Computing Practice and Programming; Computers and Communication Devices; and Community, Global and Ethical Impacts) across two levels. The first level "Computer Science and Me" is aimed at grades K–6 and the second level "Computer Science and Community" is aimed at grades 6–9. Level 1 introduces elementary school students to foundational concepts in CS by integrating basic skills in technology with simple ideas about computational thinking. At Level 2, students begin using computational thinking as a problem-solving tool.

FIVE REASONS WHY CS LEARNING IS CRITICAL FOR K–8 STUDENTS

REASON #1: Thinking is Good for Thinking. We know that students at an early age are capable of thinking algorithmically. They can apply sequencing, analysis, and testing in a number of computational settings to prescribe an action or a behavior in space and time that they design for a computational agent. As a general problem-solving strategy, this ability to understand and describe processes in time and space (algorithmic thinking) becomes a strategy that students can add to their general problem-solving toolkit. This thinking skill is not limited to solving problems in one domain; it is applicable in many domains. Furthermore, students can develop habits of mind and perseverance in problem-solving that can last a lifetime.

REASON #2: Sustaining the Next Generation of Creators and Innovators. We need to support the development of students as the next generation of creators and innovators. Watching young children play with blocks and manipulatives, we see that they are capable of, and engaged in, creative play, innovation, and exploration. Creating artifacts with computational tools is an extension of this creative play. As K–6 students are exposed to technologies through entertainment, communication, and social applications, it is important that they see themselves as more than end-users or consumers. Computing power and the skills to harness this power are the "engines of innovation." Maintaining creative expression from early experiences as creators and innovators using technology to the creation of new technologies is vital.

REASON #3: Empowering Students to Change the World. Bridging from early experiences as creators, teachers can empower students to apply their creativity and skills to solve problems. In upper elementary school, students can begin to experience computational thinking as a means of addressing community-relevant issues. The learning experiences created from these standards can be made relevant to the students and promote their perceptions of themselves as proactive and empowered problem-solvers within their community and innovators capable of changing the world.

REASON #4: Preparing Students for Future Endeavors. Early exposure to the five strands in the *CSTA K–12 Computer Science Standards* significantly impacts students' progress towards higher-level CS classes and programs. We all come to identify ourselves with what we make time for: our hobbies, interests, and priorities. If CS activities and programs are not offered to young learners, they will have no opportunity to develop a deep sense of their own technological potential. As students begin to master fundamental CS concepts and practices, they learn that these concepts and practices empower them to create innovations, tools, and applications.

REASON #5: Collaboration, Communication, and Teamwork—Key 21st Century Skills. Students working in teams often encounter multiple perspectives and create multiple solutions. Collaborative problem solving prepares students to work in teams and builds supportive partnerships.

For all of these reasons, CS is critical to student learning from the very beginnings of their school experience.

Teaching Kids Programming

Understanding the Intentional Method

Lynn Langit

Teaching Kids Programming (TKP) is a non-profit organization of volunteer programmers who have developed a framework designed specifically for introducing basic programming to children ages 10 and older. The only prerequisite is that the children have basic keyboarding skills. The framework consists of lessons ("recipes") and video-based guidelines for teachers. All materials are free at: *www.TeachingKidsProgramming.org*.

TKP teaching is based on the "Intentional Method." The TKP Intentional Method means teaching by guiding pairs of children to translate English comments—the Intention— into executable code. TKP uses Microsoft SmallBasic, plus our open source extensions called SmallBasicFun. We have also developed Intentional teaching materials for Java, T-SQL, and for Microsoft Kodu (visual programming). Additionally, TKP methods are based on Agile programming techniques (in particular XP). These include the use of core Agile practices in the delivery of TKP courses.

- Pair Programming: both the students and the teachers work in pairs to learn and teach TKP material.
- No Big Upfront: teachers are advised how to guide the student pairs to write and execute their first program within five minutes of the start of each TKP class.
- Test-driven Development: courseware is written so that students can be guided to translate one line of English into one line of code and then to execute the result. This is a type of visual test-driven development.
- Sustainable Pace: careful attention is paid to the pace of the class. Students (within each pair) rotate tasks either on task completion or on a regular time interval (such as five minutes). Pairs are also switched at the end of each lesson.
- Rapid Feedback: in addition to the immediate visual feedback that the students get after they run each line of code in the recipe, we use proctors in the classroom to keep the pairs on pace. In addition to live proctors, the courseware includes a Virtual Proctor, which provides visual feedback (a screenshot of each successful execute).
- Craftsmanship: each recipe contains several sections, so that students can progressively master concepts, APIs, and tools before being introduced to new information.

TKP coursework is designed to be modular. Each recipe is created to teach up to three core programming concepts. Examples of such concepts are objects, properties, keywords, for-loops, and events. The courseware is designed to appeal to both genders and is currently being used in 15 U.S. states and 10 countries.

There are up to five sections to each recipe. Each section takes from 15–30 minutes to teach.

- Recipe: pairs are guided to create an output starting with Logo-like drawings.
- Recap: instructors re-do (recode) the recipe so the students have time to understand what they have learned.
- Variation: students do refactoring and then change their code.
- Quiz: pairs complete a short quiz in the IDE using the concepts just learned.
- Enrichment: pairs work to further reinforce concepts learned.

LEARN MORE:

Teaching Kids Programming ***www.TeachingKidsProgramming.org***
Small Basic ***smallbasic.com***
Kodu ***fuse.microsoft.com/page/kodu***

Exploring *Thinking Myself*

Kiki Prottsman

There is no colloquialism that says, "If at first you don't succeed, give up and have a sandwich," and it is unlikely that such a phrase will ever become popular, as long as we continue to value innovation and discovery. The idea of "working until you succeed" is embedded in the concepts of scientific methodology, entrepreneurship, and of course, computer science (CS). Unfortunately, the emphasis on graded assessments has made it difficult to encourage children to learn through failure.

Thinking Myself (*games.thinkingmyself.com*) was created to encourage the try-and-try-again model. The site is an educational adventure that encourages children to navigate individually through lessons, while reading and playing games as they go. The freedom of trial-and-error learning is encouraged and the pressure to be perfect is removed from the process. It was developed using preferences and learning styles found to be desirable to girls over eight years of age.

The lessons of *Thinking Myself* encourage learning through exploration as they guide learners through exploration of computational thinking concepts. The concepts are simplified and condensed so that young children can easily absorb the lessons without adult explication. After an initial warm-up activity in which students explore an undirected challenge called the "Machine without Instructions," they are guided through exercises on decomposition, patterns, abstraction, and algorithms.

These exercises use computational thinking terminology. Though it may seem inconsistent to simplify the lesson while maintaining the vocabulary, the decision was made to reduce the fear of words so that children would not automatically associate complex words with difficult concepts. *Thinking Myself* aims to tame words that many adults find cringe-worthy.

In the first lesson, "Decompose," students learn the word decomposition and work to break problems into smaller pieces. After some instruction and an example, students decompose the problem in a sorting game.

In the "Patterns" lesson, students learn about patterns while searching through triangles in a tangram game.

The "Abstraction" lesson introduces ideas from CS, including inputs, outputs, and variables. The Abstraction game combines user input to create a specified output.

In the "Algorithms" lesson, students explore an algorithm in the form of a recipe and are then challenged to follow directions for a pirate treasure adventure.

While *Thinking Myself* is entertaining as a digital resource, all of the concepts can be presented in the classroom with the free and reproducible materials at: *games.thinkingmyself.com/analog*.

Game Programming with Alice

A Series of Graduated Challenges

Shannon Campe, Linda Werner, and Jill Denner

With the growing availability of child-friendly game programming environments, and the apparent potential of this approach for engaging students in computer science (CS) concepts, many teachers have started to integrate game programming into their classes. But there is little guidance on how to most effectively and efficiently teach students what they need to know to program a game.

Over the last ten years, with funding from the National Science Foundation, we have held game programming classes for middle school students. Our most recent project focused on how middle school students think computationally through programming games in Alice. Over 325 students (working solo or in a pair) learned to program games using Alice as part of in-school and after-school elective technology courses in seven California central-coast schools, resulting in 231 game projects.

We utilize a *use-modify-create* model for game design (Lee, 2011).

- **Use:** students play Alice games made by other students and adults and complete Alice-published tutorials on the programming environment and basic operations.
- **Modify:** students follow step-by-step, self-paced written instructions called "challenges" that teach a previously unlearned Alice programming feature and reinforce concepts learned in previous challenges.
- **Create:** students design their own unique game.

We created 17 "graduated" challenges that start with basic Alice programming and scene-design operations and move to more complex examples using combinations of operations. We found that this progression of challenges allows students to move at their own pace and learn a large number of Alice skills to draw on during the "Create" stage.

Students complete the first 11 challenges before starting to program their self-designed games. Within the first six challenges, students practice a variety of Alice features needed to create an interactive game:

- scene setup;
- adding and modifying objects (including properties to make objects invisible);
- adding, creating, copying, and organizing methods;

- using the parallel programming instruction ("Do together"); and
- creating and using events.

The third challenge in the series introduces no new skills; students experiment with Alice features they learned in the first two challenges and create their own Alice world. This challenge provides time to play with Alice and is particularly appealing to students who resist following lengthy written directions.

The next subset of five challenges focuses on specific programming skills for creating games with Alice. These later challenges are more complex and include:

- setting camera viewpoint within and between scenes;
- functions (e.g., use of proximity to create "collision detection");
- setting up and moving to different scenes;
- variables (e.g., counters), if/else conditionals (e.g., used for game player questions); and
- parameters.

Students often identified additional skills they wanted to learn once they started designing and programming their own games.

We created a group of six "bonus" challenges to address additional skills that students requested. These challenges cover: creating and importing billboard objects for instructions, sound, light effects, vehicle property (used to make objects move together), lists, loops, and more variable use (i.e., timers). This set of challenges is used to keep the faster students engaged while the others finish the required challenges, so all can move on to the game-design step together.

All of the challenge directions include features we found to be essential for students' understanding. For example, each has a list of learning objectives specific to that challenge, asks students to preview a file showing what their completed challenge file should look like, and includes reminders to save and test their files periodically; some include starter Alice worlds upon which to build.

The starter files include objects and Alice programming features students have learned previously and assist students in focusing on learning a new programming feature. The challenges also include screen shots of the working file whenever a new feature or operation is introduced, and less detailed instructions for operations covered in previous challenges. Challenge directions and supporting Alice worlds can be downloaded for use at the CSTA Source repository (*csta.acm.org/WebRepository/WebRepository.html*).

We are currently analyzing games and will compare them to the number and type of challenges completed to determine if there is a correlation between the number of challenges completed and the complexity of finished projects. We are curious to discover if students used only those Alice features from the challenges or if they used Alice features we did not cover in the challenges. We have published and reported upon some of our research findings at SIGCSE 2012 (Werner, 2012). We welcome your comments and questions. Please send any inquires to Shannon Campe at: *shannonc@etr.org*.

LEARN MORE:

Welcome to iGame ***psweb.etr.org/igame/demo/index.cfm***

Lee, L., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., & Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads 2*(1), 32-37.

Werner, L., Campe, S., & Denner, J. (2012). Children learning computer science concepts via Alice game-programming. *Proceedings of the Special Interest Group on Computer Science Education* (SIGCSE).

Teaching Java to Fifth Graders

Vic Wintriss and Aaron VonderHaar

Conventional wisdom holds that ten-year-olds have not yet developed abstract thinking abilities sufficiently to enable them to write computer programs—especially in Java. Our experience has been quite different. At Wintriss Technical Schools, a small, non-profit, after-school program, Java professionals use kid-appropriate terminology, training aids, and curriculum to teach children to write computer games.

Our teaching methods contribute to student success. Unlike classes that try to teach with lectures and boring exercises, our students write complete games from the very first day. Ten volunteer professionals mentor the students to create games such as Hi-Lo, Pong, Tic-Tac-Toe, Memory, and Asteroids. An autonomous robot competition further tests the skills and imagination of students.

Java professionals are willing to volunteer to teach students when they learn that the teaching environment is not a classroom of thirty students. Groups of about three students meet for two hours, once a week, in a classroom of four Apple iMacs. Student-teacher relationship building is important and teachers find that the strong relationships formed in this setting are very rewarding and an important feature of the program. All of the volunteers are screened via LiveScan.

My job is to teach the teachers kid-appropriate teaching methods. Our volunteers avoid using scary, technical terminology as much as possible before beginning students learn that they can actually write a sequence of code that produces fun output.

Teachers describe how to make a blueprint (Java Class) "real" by using the Java keyword "new", instead of talking about instantiating a Class to create an object of the Class. Teachers also help students conceptualize programming concepts; methods have a "mouth" (the space between the parentheses in a method header), and you feed input parameters into a method's mouth. Methods also have "guts" (the area between the opening and closing curly brace in a method body) where the code for this method goes. The method return type specifies what comes out of the guts!

We also introduce students to real-world contexts for many of the math concepts they are learning in school by using the x-y coordinate system, trigonometry, and simple physics simulations in their games. Because our teachers are Java professionals, they are not limited to pre-scripted projects. Their programming expertise gives them the flexibility to adjust the games on-the-fly.

Five years of teaching Java to students from fifth grade through high school, have enabled us to develop an extensive collection of projects. We use NetBeans as the IDE and Subversion for the source control program. All student work is saved on a school server and accessible from home.

So far, four students have passed the Advanced Placement Computer Science (AP CS) exam; one in the eighth grade, one in the ninth grade, and two in the tenth grade. Students who pass the exam have had the incredible opportunity of being placed in paid, summer intern jobs. Our oldest student graduated from high school this year and is now attending Stanford as a CS major. Most importantly, the students seem to genuinely enjoy the new-found freedom of expression that comes with writing computer programs.

Supporting Kids' Explorations of Computation

Karen Brennan and Mitchel Resnick

Many kids today are comfortable interacting with computers, able to use standard applications, and search for information online. But how can we help them go further, so that they can express themselves with computation and learn important computational ideas?

By learning to program, kids can develop a deeper understanding of, and greater fluency with, computation. But there is a problem: traditional programming languages are difficult for kids, with challenging syntax and lack of immediate feedback. To support kids' explorations of computation through programming, our research group—*Lifelong Kindergarten* at the MIT Media Lab—developed Scratch, a graphical programming language that makes it easy for kids to create their own interactive media. To create programs in Scratch, you simply snap together programming-instruction blocks, just as you might snap together LEGO bricks or puzzle pieces. You can instantly see the effects of instructions whether you click on an individual block or an entire stack of connected blocks. These qualities minimize the demands of mastering syntax and support immediate feedback and incremental design.

Another major component of making programming accessible (and appealing) is helping kids make connections to their personal passions and to other young creators. To do that, Scratch provides kids with not just an authoring environment but also an online community, where tens of thousands of young people (mostly ages seven to seventeen) have contributed more than 2.2 million projects since May 2007. These projects represent a wide variety of interests and passions, from role-playing games to interactive musical interfaces to simulated worlds, with creators using Scratch to make projects about their favorite music, books, games, hobbies, and academic subjects. The online community serves as a large library of creative inspiration and as a space for creative collaboration. Community members give each other feedback on work and team up to create projects bigger than they could have created on their own.

Creating with Scratch occurs in a variety of settings: in and out of school, across disciplines and ages, from kindergarten to college. Out of school, many kids find Scratch on their own or through family and friends. For example, Tim, a 9-year-old boy who loves to play *The Sims,* wanted to make his own simulations. He found Scratch by searching online and learned Scratch by downloading projects from the online community to see how they worked. He made his own Scratch-based people simulation, but created a variety of other projects, including a stop-motion animation featuring a plastic shark, an animated music video of his favorite pop song, and a greeting card for his grandmother. Tim taught his mother how to create Scratch projects, and they collaborated on a role-playing game. Both mother and son were excited to get positive feedback on their project after they posted it online.

We also see these connections to kids' interests happening in schools. A fifth-grade teacher and her students selected Aesop's fables, brought them to life by programming animated stories, and shared their projects online. A second-grade teacher and his students were studying animal tracks. They used Scratch to program simulations of their favorite animals leaving tracks in freshly fallen snow and hosted parents and friends for a Scratch open house. A computing teacher and fourth-grade visual arts teacher worked with students to create interactive art in the style of a visual artist they were studying. Students incorporated images of themselves in the art and the class invited visitors to help with the works-in-progress. A seventh-grade computer studies class was learning introductory programming through game design and invited third-graders to serve as game testers for their maze projects.

Scratch is being used in a variety of settings to support kids' explorations of computational ideas. With an easy-to-use interface and an emphasis on connecting with interests and people, we see a wide variety of kids deeply engaged in computational expression, including kids who might never have imagined themselves as designers of interactive media.

ScratchEd (*scratched.media.mit.edu*) is an online community for Scratch educators, where members share stories, exchange resources, ask and answer questions, connect with other educators, and find out about upcoming events. You can view videos of teachers describing their experiences of working with Scratch in the classroom or download the new Scratch curriculum guide for activity ideas. Teachers can also participate in monthly meetups and workshops hosted at the MIT Media Lab, free monthly webinars, and the biennial Scratch conference held at MIT (July 25–28, in 2012).

LEARN MORE:

Scratch online community ***scratch.mit.edu***
ScratchEd online community ***scratched.media.mit.edu***
ScratchEd updates via Twitter ***twitter.com/ScratchEdTeam***

Logo: A Language for All Ages

Michael Tempel

Logo is most often thought of as a computer environment for young children with an emphasis on graphics. But it is much more. Logo is a sophisticated programming language, a dialect of LISP that can be used by learners of all ages in a wide variety of ways.

A guiding principle of Logo development has been that it should have a "low threshold" and a "high ceiling." You should be able to enter the Logo room easily with no hurdles to jump over and no big step up. Then, once inside, you should be able to move seamlessly from simple explorations to complex projects.

A second goal is that the Logo room should have "wide walls." That is, people with different interests, tastes, and learning styles should all be comfortable. There should be a variety of domains in which to develop projects.

In the 45 years since Logo began, the room has gotten bigger. The threshold is lower, the ceiling is higher, and the walls have moved outward. To see how this has been happening we can begin with Logo as it was known to most people in the early 1980s when it emerged from the research environment at MIT and found its way into schools and homes. The most popular versions of Logo at that time were for the Apple][. The turtle geometry component was by far the most widely used.

Young children could begin exploring shapes and write procedures to draw them. Simple shapes could be combined into more elaborate designs. Older students could use the turtle for complex mathematical explorations.

Turtle geometry is just one domain for Logo explorations and projects. During the early years of development, Logo was used in many areas, including music, robotics, and language. In fact, the first versions of Logo had no turtle. The name Logo, which means "word" in Greek, was chosen to emphasize that the language was well-suited to working with words and sentences in contrast to the numeric focus of most programming languages at the time. Over the past 30 years, hundreds of versions of Logo have been developed with increasingly diverse capabilities.

ANIMATIONS AND GAMES

While turtles were crawling around on Apple][screens, personal computers that doubled as game machines—such as the Atari 800—supported colorful multiple turtles, also called "sprites," that could wear different costumes and be set in motion. Animations and video games emerged as the favored projects on these machines. This functionality is now standard in many current versions of Logo, including MicroWorlds and Scratch. These modern implementations also include drawing tools and allow the importing of different kinds of media, including images, video, sound, and music.

ROBOTICS

In the mid-1980s, work on robotics versions of Logo was underway at the MIT Media Lab. Logo programs received information from light, touch, and other sensors, and activated motors and lights. LEGO TC Logo and Control Lab were widely used products that grew out of this research.

In the early 1990s work began on Programmable Bricks. The Brick, which you could hold in your hand, had a microprocessor inside. As with the earlier Logo robotics environments, the Programmable Bricks worked with sensors, motors, and lights. But now, with a downloaded program the Brick could be disconnected from the computer and move on its own, as part of a vehicle exploring its environment, for example. The RCX and NXT from LEGO grew out of this project, along with several types of Crickets, which are smaller programmable bricks.

MODELING AND SIMULATION

Mitchel Resnick's 1994 book *Turtles, Termites, and Traffic Jams* described Star Logo, a system he developed to allow exploration and modeling of decentralized systems. Star Logo has thousands of turtles that can be programmed to interact with each other and with patches of background. One can simulate the emergent behavior of, for example, a termite or ant colony, or explore how traffic jams form or forest fires spread. A similar program derived from StarLogo is NetLogo.

BLOCKS PROGRAMMING

In 2006 a new version of Logo called Scratch was developed by the Lifelong Kindergarten Group at the MIT Media Lab. It was designed to enable creation of games, animations, and multimedia projects. The key difference from earlier versions of Logo with similar capabilities was the use of "Blocks Programming." Instead of lines of text, programs are constructed by snapping together blocks that fit into one another like jigsaw puzzle pieces. Program structure is represented visually. A great advantage of Blocks Programming is that it is almost impossible to make the kinds of syntax errors that are common with text programs, those that result from typos and incorrect punctuation. The different types of blocks are of different shapes and fit only where they are syntactically appropriate.

Blocks Programming did not originate with Scratch. It was first developed in 1995 as Logo Blocks for the Programmable Brick, which is still in use today. However, the immense popularity of Scratch has brought Blocks Programming to the forefront in the educational technology community.

Another important aspect of Scratch is the community that has grown around it. The Scratch website has over a million members and almost two and a half million projects that have been posted. The forums for discussion and sharing are part of the Scratch culture. Many projects are developed by remixing those that were previously posted.

Logo is widely used at the elementary and middle school level where it can be integrated into many subjects. This is less the case at the secondary and college levels, with the exception of StarLogo and NetLogo. Concepts that students encounter in Logo-type courses, including structured programming, algorithms, data types, and objects, support learning any programming language. After 45 years, Logo remains a language for learning for people of all ages and accommodates an ever widening variety of interests and learning styles.

LEARN MORE:

Logo Foundation Web Site ***www.logofoundation.org***
Logo Tree Project ***www.elica.net/download/papers/LogoTreeProject.pdf***
Logo Update ***el.media.mit.edu/logo-foundation/pubs/logoupdate***
Cricket ***gleasonresearch.com/prod.php?sku=SUPERCX***
PICO Cricket ***www.picocricket.com***
NetLogo ***ccl.northwestern.edu/netlogo***
StarLogo ***education.mit.edu/starlogo***

SPECIAL ISSUE CS K–8

implementation

Integrating CS into Middle School Projects

Adventures in Alice Programming

Susan Rodger

Middle school students use computers in many aspects of their daily lives, yet the majority of them do not experience computer science (CS) in school. The Adventures in Alice Programming project (*www.cs.duke.edu/csed/alice/aliceInSchools*) encourages teachers in every discipline to integrate computer programming as a tool for students to use in class projects. Alice 3D (*alice.org*) with its drag-and-drop interface, makes it easy for students to create simple animations and has great potential for exciting students about CS in the same way that biology, chemistry, and physics experiments can excite students about science.

To encourage the use of Alice in schools, we have developed free curriculum materials which are available on the project website. They include four types of tutorials and many sample worlds to fit different disciplines. Getting-started resources range from short introductions to four-part, in-depth tutorials. Animation tutorials include techniques for moving the camera and adjusting the lighting. CS concept tutorials show students how to make a decision using an "if" statement.

For language arts, a book report tutorial shows students how to animate elements normally found in a book report. Unlike a static poster, the Alice book report may include one or more animations or interactive quiz questions. A sample history tutorial about a bridge shows how to implement and switch between several scenes. A language tutorial shows how to build an animation to teach Spanish. In one such tutorial, a girl is baking and giving instructions in Spanish for the user to place particular ingredients into the mixing bowl. A science tutorial shows how to build a helium molecule using sphere shapes.

Some tutorials teach about building games. There are a large number of tutorials for middle school math games, including a game for testing inequalities, a game for multiplication facts, a game on plotting points, and many more.

Duke University will be offering summer Alice workshops for teachers for the next four years. Undergraduate students help run the workshops and work with teachers. The target audience is North Carolina middle school and high school teachers who are not already teaching programming, including teachers of language arts, math, science, history, music, art, foreign language, multimedia/technology, and business. Collaborators on our grant will also be running workshops in South Carolina and Mississippi for the next four years.

The project is supported by NSF ITEST grants, IBM Faculty Awards, and CRA-W DREU awards, and currently involves investigators at Duke University, College of Charleston, Columbia College, University of Mississippi, Stanford University, and Carnegie Mellon University.

Introductory CS in the K–3 Classroom

Patrice Gans

Where is it written that computer science (CS) is a learning experience reserved for middle and high school students? Fortunately, some schools across the country have made CS a priority and are currently nurturing the art and joy of CS in primary school students. The view of students as "creators" rather than just "consumers" is firmly rooted in my curriculum at Fraser-Woods in Newtown, CT.

Over 26 years ago, the book *Mindstorms: Children, Computers and Powerful Ideas* by MIT Professor Seymour Papert introduced the educational community to the concept that computers could serve as an environment for "thinking about thinking" and, simultaneously, developed the first educational programming language, Logo, designed specifically for elementary students. Papert envisioned a learning environment where children would program computers to acquire "a sense of mastery over a piece of the most modern and powerful technology." Students would become masters of technology instead of just users.

As the technology boom continues to invade our homes and schools, it is increasingly important for teachers to embrace Papert's vision. I introduce my students to rudimentary CS principles beginning in kindergarten with the application Ladybug Mazes and Ladybug Leaf from the Library of Virtual Manipulatives. The applications are similar to Logo; students create instructions (programs) to navigate a ladybug across the computer screen by selecting the correct icon to move the ladybug in the required direction. Steps are recorded in a plan window so that the students can visualize how each step contributes to the ladybug's overall movement.

After the students became proficient with the movement commands, they begin work with Ladybug Leaf in which they use manipulatives to create geometric shapes. They start by creating programs to draw squares and a rectangles and advance to creating parallelograms, triangles, rhombuses, octagons, and hexagons. Finally, the students write programs to create their own designs.

The Ladybug Leaf and Ladybug Maze applications provide an informal introduction to CS without any of the complications of a computer language or syntax. Initially, my students tackle the challenge of navigating the maze by trial and error, mixed with a fair amount of frustration. However, as we work together to understand the commands, the students are able to efficiently move the ladybug through the maze. At this point, their frustration turns to elation. My second and third graders have similar experiences with Ladybug Leaf. By the end of the unit, they are developing programs which enable them to create beautiful geometric patterns. Developing code is no longer an enigma. They are proud to call themselves computer scientists.

Active Learning Ideas for K–5

Karen North

Computer science (CS) is a design methodology that teaches students to analyze and design solutions to problems. It doesn't matter if the desired goal is writing a persuasive essay or writing a mobile app—the skills from CS translate into creating the solution. In the elementary school I focus on the following CS skills that are needed in every academic area:

- Designing solutions to problems
  - Writing a sequence of steps
  - Solving puzzles
- Following directions
- Comparing and contrasting
  - Looking for patterns
  - Paying attention to detail
- Developing spatial skills

Connecting the topics children love with CS concepts is a winning combination. Teach students to control their digital devices and include lessons on binary numbers. Operating robots also provides a natural occasion to talk about algorithms. Students' love of pets and animals provides a wonderful opportunity to introduce them to careers in bio-informatics. The superhero devotees can explore digital forensics. As teachers, we can start early to connect students' interests to the world of CS. Ideas and resources abound so I will focus on a few specific activities I have found to be engaging and easy to implement.

BINARY CLOCK

I use a binary timer to visually introduce binary numbers and counting. My classroom binary clock challenges students to convert a typical 12-hour clock to telling time in binary. I use the "fold-a-book," normally used to teach fractions, to create a booklet for drill and practice. Using an unfamiliar time recording system helps students understand the passage of time and how time recording devices work. Young kids can make patterns with zeros and ones, older learners can explore on their own by searching machine language or binary numbers. You can also download the binary clock and add it to your desktop (*www.sb-software.com/binaryclock*). More details and lesson plans can be found on the IEEE TryEngineering website (*tryengineering.org*).

BEE-BOT

One of my favorite resources is the Bee-Bot programmable floor robot. It has six simple commands and no computer interface is required. Students learn to sequence steps, a skill that is essential for processing information in any area. After the basic six commands are mastered, students can add new CS concepts and commands for distance, angle, procedures, and loops using the Pro-Bot robotic car.

Bee-Bot tools reinforce spatial learning, as well. Students create a robotic spelling bee in which they must program the Bee-Bot robot to move from letter to letter on a floor grid to spell a word. With Bee-Bots, students can practice mathematical number-line concepts through movement. First, students simulate the program with their own bodies by walking the sequence of steps. Later, the sequence can be programmed into the Bee-Bot. Add music, and students can create their own dance. The Dancing with Bees video can be seen at: *cscurriculum.shutterfly.com/35*. Learn more about Bee-Bot and KinderLogo at: *www.terrapinlogo.com*.

WESCHEME

My favorite programming resource for this age group is WeScheme. The cloud environment implements a problem-solving methodology that can be applied to any problem. A popular project involves designing a water cistern. In this project, students design their own solutions in WeScheme. This activity requires students to apply skills in media computation and geometry. WeScheme is so simple that young children can use it. Advanced lessons on programming with graphics can be found at: *teachertech.rice.edu/Participants/knorth/Scheme/web Scheme.html*. The Programming with Graphics video can be downloaded at: *cscurriculum.shutterfly.com/36*.

Computer scientists will be vital to solving the problems of the world. That is why it is imperative to begin producing those scientists today in elementary schools. This will require a change in attitude about the role of CS education for young children and allocation of time and resources. Bringing about these changes will require determination and rigorous work; and as my students say, "this is hard, but fun."

Media Mania with Kodu

Elisa Rossi

Media Mania is an elective class for seventh- and eighth-grade students in Southlake, TX, a city located between Fort Worth and Dallas. Carroll Middle school offers three technology classes teaching skills ranging from keyboarding to video game design. Media Mania includes photo editing, movie making, and video game design using Kodu, a 3D visual programming environment in which students create their own visual worlds with terrains, characters, and complex behaviors.

I originally used Kodu as an extension activity, letting students use it independently during their free time and provided little to no instruction. Students were eager to learn how to make exciting things happen and asked questions like, "How do I make the Kodu character disappear?" or "Can I make it lose points after so much time has passed?" Because of their eagerness and motivation to learn more, I decided to include it as a new unit of study. My students have created maze games, side-scrolling games, adventure games, racing games, and many more. Most recently, students have figured out a variety of ways to create an appearance of multiple-level games.

Kodu (*fuse.microsoft.com/page/kodu*) enables kids, typically ages eight and above, to create games on the PC and XBox through a simple visual programming language. The Kodu Classroom Kit is a set of lessons and activities that includes plans, tutorials, and a series of introductory videos. In the Kodu community (*KoduGameLab.com*), students can discover games created by others and share their own games. Kodu and the supporting learning resources are free. Each month a "gamejam" contest provides teachers and students the opportunity to showcase their creativity in solving problems with Kodu (*koduwebdnn.cloud app.net/Home/tabid/55/forumid/6/postid/486/scope/posts/Default.aspx*).

For projects, I typically assign a game genre and basic game requirements such as including timers, scoring, or cause-and-effect scenarios. The students are required to create a backstory and specific objectives for each game. I also provide students with a set of the games created in class so they can enjoy them or continue working on them at home. Most of my students have downloaded Kodu at home by now, and come in talking about the games they are creating. They really seem to enjoy it and are disappointed when it is time to move on to the next unit.

Kodu is a great starting point for learning about game design and basic computer science concepts. The sentence-like structure of the language makes it is easy for students to comprehend the logical cause-and-effect relationship of commands. I find that students who understand the basics are able to transfer that knowledge to planning and programming in other languages.

KODU MAZE GAME ACTIVITY

Objective: Create a Kodu maze game

Requirements:

- A backstory
- Instructions
- Two main characters (one free roaming, one user-controlled)
- Two levels
- Power-ups
- A timed activity
- Scoring

Planning:

- What is the objective of the game?
- How will my characters gain and lose points?
- Is there a way for characters to obtain special powers?
- How will the characters progress from one level to the next?

MAZE GAME SELF-ASSESSMENT

Title: ____

Creator: ____

Objective of game: ____

____

List all active Objects in game: ____

____

List all inactive Objects in the game: ____

____

Are sounds or music included? If so, what is the purpose of the sound and/or music?

____

____

What are the best features of the game? ____

____

____

What would you change to improve the game? ____

____

____

List three programming techniques you learned in this project and describe how they were used.

____

____

10-Year-Old Detectives

Mark Dorling

The Digital Schoolhouse (DSH) is a transition project established by Langley Grammar School (LGS) in the United Kingdom to offer predominately Year 6 (10-year-old) pupils from local primary schools the opportunity to visit LGS for a day of learning in a dedicated Information and Communications Technology (ICT) and computing environment.

To accommodate the different learning styles, the DSH has developed a variety of audio, visual, and kinesthetic teaching activities. For example, pupils perform a human-database role play using SQL syntax to structure questions and then move around the classroom to stand in groups and arrange themselves into an order depending on the answer. Higher-order questioning is also used to relate concepts to students' "real world" understanding. This enables pupils to better grasp the skills and concepts being taught, and maximizes learning.

This year, the DSH has introduced a new lesson called *Database Detectives* based on the book *Certain Death* by Tanya Landman. The aim of the lesson is to highlight the role of ICT and computing in the logging and analysis of the data generated in the crime scene investigation lab and to illustrate how detectives use the clues to solve crimes. Before completing the lesson, the teacher is encouraged to read the book (except the last chapter!) and complete a series of numeracy puzzles loosely based on the book. The puzzle answers provide pupils with the clues to identify the murderer.

Pupils use cloud computing technology such as Google Documents spreadsheets, to collaboratively input data about the suspects from profile cards based on the book. Pupils then perform verification on their neighbors' data entry before downloading the spreadsheet and importing it into Microsoft Access. After importing the data, pupils use the filter tools to solve the murder using the answers from the numeracy challenges and then create reports for the Court based on queries identifying the murderer.

DSH pupils work with a primary-trained teacher, who also has secondary ICT teaching and industry experience, to learn new skills and concepts with a focus on how they are deployed in secondary education, the world of work, and business. The DSH approach features a creative curriculum based on the proposed changes to the British national curriculum. The creative curriculum links have been developed through collaboration with school subject specialist teachers and the library resource center. The DSH is open two days each week. The day starts at around 10:00 am and finishes at 2:30 pm. The DSH lesson is split into three sessions lasting approximately 60 to 90 minutes.

The techniques in the DSH could have future applications in teaching pupils with special educational needs or who have missed out on good ICT teaching in their earlier schooling. The program is currently being extended to cover post-secondary education topics like relational databases.

If you would like to discuss the concept of the DSH in more detail, please contact Mark Dorling at: *dsh@lgs.slough.sch.uk*. Further reading about the project is also available from the Digital Schoolhouse (*www.digitalschoolhouse.org.uk*).

College for Kids

Cindy James

For many years, my son had attended the College for Kids program at Illinois Central College (ICC) in East Peoria, IL—the same campus I had attended. The experience was a wonderful early opportunity for him and other young students to explore career choices.

Over the years, I had always noticed that the offered technology courses filled up quickly. After witnessing the excitement and accomplishments in my K–8 technology classes in Norwood District 63, I contacted Erica Peterson, College for Kids Program Coordinator, to offer more computing courses.

The three courses I proposed were accepted. *Let's Paint* did not meet the enrollment criteria, but *Amazing PowerPoints* was a big hit and *Computer Programming with Etoys* was full during both sessions. The curricula for all three courses are available in the CSTA Source web repository (*csta.acm.org/Web Repository/WebRepository.html*).

In *Amazing PowerPoints*, students explored a wide variety of features, including Custom Animations and Motion Paths. Students left eager to create presentations to show-off their new skills to "amaze" their teachers and peers.

"Awesome" describes the *Computer Programming with Etoys* class; students were completely engaged and begging for more each day. Etoys is a unique authoring environment that encourages both creativity and problem-solving. Throughout the class we discussed the value of problem-solving skills and how, just as in life, the more skills you have in your toolbox, the more able you are to solve the problems that come your way. That is the answer to the age old question: "Why do I have to learn all this stuff?"

The lesson plans for eight days of Etoys activities are included in this publication and are available, along with the others, in the CSTA Source web repository. Student projects can be viewed on the school website (*teacherweb.com/IL/Norwood63/MrsJamesTechnology/apt1.aspx*) and on EtoysIllinois (*www.etoysillinois.org*).

An 8-Day Plan with Etoys

Cindy James

Editor's Note: These lessons were originally delivered in a summer College for Kids program. Each "day" is the equivalent of two 50-minute class periods. For a description of College for Kids refer to the "College for Kids" article in this publication.

OBJECTIVES:

Students will learn basic computer science and game design concepts through the creation of an Etoys animation. Students will develop problem-solving and logical-thinking skills.

Supplies:

- Etoys program for teacher and student computers (*etoysillinois.org/download*)
- Tutorial videos (*waveplace.com/resources/tutorials*)
- Internet connectivity
- Projector for the teacher computer

DAY 1: Introduction to Programming and Etoys

Activity: Show projects on Etoys Illinois Library Collections (*www.etoysillinois.org*).
Introduce terminology (Project, Supplies, Sketch, Halo, Sprite, Script, Stage and Background).
Instruct students to open the Etoys program and follow along with the Basic Etoys Lesson videos 01–04 (*waveplace.com/resources/tutorials*).

Learning: The ability to create and save a Project, use the Supply Box, create a Sketch, and use a Halo to manipulate the Sketch; knowledge of basic programming and animation terminology.

DAY 2: Etoys Scripts & Viewers

Activity: Instruct students to follow the Basic Etoys Lesson videos 05–06 to begin their first Projects and write Scripts.

Learning: Ability to create a simple animation.

DAY 3 & 4: Etoys Variables

Activity: Instruct students to follow the Basic Etoys Lesson video 07–08 to create variables in new or existing Etoys Projects and to test the accuracy of their programming.

Learning: An understanding of, and ability to use, variables in an Etoys Project; testing skills for use in the next lessons involving animation and game creation.

DAY 5: Get Animated with Etoys!

Activity: Instruct students to follow the Basic Etoys Lesson video 09 to create and experiment with costumes for the animations in new or existing Etoys projects.

Learning: An understanding of animation concepts; the ability to create costumes and animations.

DAY 6: Game Creation with Etoys

Activity: Instruct students to follow the Basic Etoys Lesson video 10 to create animated games using new or existing Etoys projects.

Learning: The ability to create a simple animated game.

DAY 7: Animated Stories

Activities: View examples of animated Stories on the Etoys Illinois website. Students will create simple animated Stories. (*etoysillinois.org/library.php?tags=Stories*)

Learning: Students will learn how to create animated Stories.

DAY 8: Sharing Student Work

Activity: Allow students time to complete storybook projects to post on Etoys Illinois. (Student work is identified by first name and grade level only.)

Encourage students to download Etoys (free) at home for continued learning and skill development.

Learning: Increased competency with Etoys; continued development of problem-solving and logical thinking skills.

Press Play

An After-school CS Program

Rebecca Dovi

Most middle school students fall into two after-school categories: latchkey kids who spend all afternoon zoning at home in front of a computer game console, or daycare kids who spend all afternoon away from home in an institutional setting doing arts and crafts projects.

While parents of middle school students see these as less than adequate options, computer science (CS) teachers should see a golden opportunity.

In 2009, I started Press Play, an after-school CS club for middle school students which, in reality, is operated by my high school CS students. The club is the perfect combination of creativity and engaging technology but without the less desirable effects of other after-school options. To call the camp a success is a huge understatement.

We operate the Press Play camp once a week for four weeks. Each session lasts two hours, which fills that critical span of time between the end of the school day and the end of most parents' work days. Transportation in our district is simple; students ride the bus from a middle school directly to the high school camp location.

We use Alice and Scratch; both are simple programming environments built with fun and function in mind. Students create their own digital storytelling and simple, but fun game designs.

In addition to game design, students learn about robotics. LEGO Mindstorm products are a tried and true way of engaging students in both computing and in simple engineering—and in the context of Press Play, they are the perfect digital spokesmodel to convince middle school students that CS is the class to take once they enter high school.

At the end of the four-week program, each student receives a t-shirt and a USB drive with their games and stories, plus all of the software they used to make them so they can keep on "pressing play" at home.

The registration fee of $40 per student supports our high school CS program similarly to any fundraiser and the high school students who teach the middle school students earn service experience. It is impossible to overstate the value of this program as a recruiting tool.

Of the middle school students who have participated in the program over the past few years, about 40 percent take a CS course at the high school level.

By turning the Press Play into an after-school fundraiser run by my high school CS students, I've been able to use Press Play for maximum benefit to my current CS students, to my program's enrollment, and to the benefit of my future students by giving them a fun preview of what to expect from CS classes in high school.

Computer Science as an Art Form

K-5 Fine Arts

Christopher Michaud

Computer science (CS) equips students with unique skill sets similar to music and dance. The skills to model reality through numbers, methods, and functions give CS a unique place alongside music, dance, and drama as a way for humans to express creativity.

How do we teach CS to younger students? We deliver concepts and content sequentially through culturally relevant activities to create and play games, tell stories, and run simulations that equip students with the tools for digital self-expression.

Kindergarten, first-, and second-grade students experience learning through kinesthetic actives of art, dance, music, drama, and sports. TuxPaint (tuxpaint.org) or other simple painting programs allow students to express ideas in images, colors, lines, and forms. These activities develop user-interface skills, lead students to think about objects in the computer as similar to objects in their drawings, and encourage students to see technology as an expressive tool. I use painting, drawing, and text manipulation through standard office software to teach third-graders the skills of manipulating text, creating and editing images, navigating a file system, and saving and retrieving their work. I teach CS concepts to fourth- and fifth-grade students using Scratch, Lego NXT Robotics, and Python.

CS is taught in the domains of linear and event-driven sequences, methods, and models, which are introduced sequentially and built one upon the other. First, students learn to arrange directions or commands in a linear sequence. After students have mastered commands to animate, draw, and drive a robot, we begin to label command sequences that we use over and over again.

Labeling sequences or creating methods ("methoding") is the second domain. Creating methods that can be reused represents an important step in understanding how to create a computer program.

The third domain is modeling objects. Through the use of data structures, students create virtual models of objects that "live" inside the program. The picture represents the character in a game or an object in a simulation.

Scratch (*scratch.mit.edu*) provides a graphic programming environment that allows students to explore, create, and remix graphics and sounds through animations, games, and simulations. Students extend the music activities of singing, dancing, and drama onto the screen. Examples include interactive artwork and storytelling. Students use Scratch to model movement with musical selections and use colors and speed of movement to reflect the musical ideas in a selection.

Students further explore modeling concepts by creating virtual representations of musical instruments. We use Scratch to create virtual instruments such as xylophones and drums (*www.nebomusic.net/scratch xylophone.html*). Students use Lego NXT Robotics to perform dance routines and play instruments to take their sequences and programs out of the "screen" and into the real-world.

I use the Python programming language to transition my fifth-grade students from Scratch to text-based programming. The JES environment by Georgia Tech (*code.google.com/p/mediacomp-jes/*) provides a platform for students to explore Python programming by manipulating pictures and sounds along with "driving" Turtle graphic programming (*www.nebomusic.net/techlesson11-5StudentDirections.html*).

CS is a powerful tool to empower elementary students in the fine arts.

Make it Fun!

CS in Elementary School

Dan Frost

Five years ago, at the urging of my daughter's fourth-grade teacher, my wife (a Director of Information Technology) and I (a college-level computer science (CS) educator) took on the challenge of teaching CS to fourth graders and making it fun!

This approach meant minimizing lectures and maximizing hands-on activities. It meant making the class more like art or physical education, and less like math or writing. It meant that students should feel lots of success. I wanted to promote the same feelings of universal success and recognition in our CS class as was apparent in the display of the student art work on the classroom walls. The "fun principle" is particularly important for a first exposure to CS, given the not-fun reputation our field sometimes has in high school courses.

Author Marc Prensky labels today's children "digital natives," and I observed that first-hand in the fourth-grade students. The school's media center had a sufficient number of computers, the students had studied keyboarding in third grade, and they had some experience with word processing and the Web.

I decided that fun, creative, hands-on programming should be a big part of the class. I wanted a language that gave immediate feedback and supported very short, interesting programs in under 20 keystrokes. Because neither HTML nor JavaScript seemed fourth-grade friendly, I decided to create a version of BASIC targeted to elementary children and delivered as a Java applet. I created it with precisely the features I needed to work with children. Being a Java applet, it could run in the computer browser, eliminating difficult installation issues.

The resulting course was indeed a great deal of fun—for the fourth-graders and for my wife and me, also. In each session, we introduced a new programming command, the students tried it out, and then proudly showed off their accomplishments. Several times we have been told that, for some students, our class was the highlight of the week.

In addition to programming activities, students learned about computer components such as input, output, processing, and storage. Most of the students knew the term "algorithm," so we examined algorithm strategies such as guessing, computing, and trying each possibility. They explored other CS concepts and terms including branching, repeating, subroutines, and kilo- and mega- prefixes. We put the CS lessons in context through discussions of computing careers and of people who have contributed to technology developments.

The teachers and principal were excited to see the transfer potential of the CS skills students had learned to the critical-thinking and problem-solving skills taught throughout the curriculum.

Over the past five years we've taught CS to about 400 fourth-, fifth-, and sixth-grade students. More recently I created an applet version of the Logo language which is now our language of choice for fourth-graders due to its graphics capabilities. The programming languages and support material are available at: *www.csed.org*.

Filling the Pipeline from the Middle School

Constance Seiden

The Community College of Aurora (CCA) has applied for, and received, a Carl Perkins grant from the Colorado Community College System Office to increase young women's interest in technology and computer science. This grant is enabling the department to train female middle school students in new technology utilizing web-based applications to increase their interest in the field and encourage them to pursue careers in technology.

Over the past decade, there has been a steady decline in the number of women entering technology fields as reported by the UCLA Higher Education Research Institute. To address this disturbing trend, a team from the CCA Computer Science Department, alongside industry volunteers, worked with the 20 middle school girls over the course of a week to create websites and explore technology career options.

Workshops were held in November and again in April of 2012, at Merrill Middle School in the Denver Public Schools. The start of each session was led by a woman volunteer with a career in technology who described her professional experience, hurdles, and accomplishments, and motivated the students to explore the vast array of career options in technology. Following the presentations, the students used iPads and MacBooks purchased with funds from the grant to create personal websites. The workshops were outstanding successes. Students responded with, "Technology is fun," "I can go far," and "It's the future and it's not intimidating." The sites can be viewed at: *www.ccaurora.edu/programs-classes/departments/cis/merrill_MS*.

Later in the spring, a Technology Preview Day was held at Aurora West College Preparatory Academy. Forty-four eighth-grade girls attended the event. The girls were divided into two groups. While one group created websites (*www.ccaurora.edu/programs-classes/departments/cis/awp*), the other group took apart a computer, put it back together, and made it run. Afterwards, the Center for Outreach Recruitment spoke to the girls about opportunities at the Community College of Aurora.

Meet the Authors

Valerie Barr
Union College, NY
Valerie is the Chair of the Department of Computer Science where she researches software testing. She is also working on a campus-wide computation program that will engage all students in computing, regardless of their field of study.

Marina Umaschi Bers
Tufts University
Marina is an associate professor in the Department of Child Development and Computer Science and heads the Developmental Technologies research group.

Karen Brennan
MIT Media Lab
Karen is a doctoral student in the Lifelong Kindergarten group at the MIT Media Lab and leads the ScratchEd project.

Shannon Campe
Education, Training, Research (ETR) Associates
Shannon is a Project Coordinator and has taught and coordinated multiple in- and after- school technology projects for middle school girls and boys.

Jill Denner
Education, Training, Research (ETR) Associates
Jill is a Senior Research Associate. Her research interests include the role of after-school programs in engaging girls with information technology, and the role of gender and culture in the development of Latino youth.

Zachary Dodds
Harvey Mudd College, CA
Zachary is a Faculty member in the Computer Science Department at Harvey Mudd College and a developer of the *Middle-years Computer Science* (MyCS) curriculum.

Mark Dorling
Digital Schoolhouse
Mark is the Digital Schoolhouse project coordinator recognized in the Royal Society Report 2012, Board member of Computing at School, and Lecturer in the School of Education at Brunel University in the UK.

Rebecca Dovi
Hanover County, VA
Rebecca is lucky to teach in Hanover County where every high school has a full time CS teacher. In Virginia, She serves as the founding president of the CSTA Central Virginia Chapter and represents Virginia nationally in the CSTA Leadership Cohort.

Mike Erlinger
Harvey Mudd College, CA
Mike is a Faculty member in the Computer Science Department at Harvey Mudd College and a developer of the *Middle-years Computer Science* (MyCS) curriculum.

Dan Frost
University of California Irvine
Dan is a lecturer in the Informatics and Computer Science departments at UC Irvine. He also teaches at the grade school and high school levels as often as possible.

Patrice Gans
Fraser-Woods, Newtown, CT
Patrice teaches technology to students K–8. She also teaches Scratch in a summer program at Naugatuck Valley Community Technical College.

Cindy James
Norwood District, IL
Cindy teaches CS & technology for Norwood District 63 (K–8) and in the College for Kids program at Illinois Central College in East Peoria, IL. Prior to teaching, she spent 30 years working in business focusing on efficiency and technology in the workplace.

Lynn Langit
Teaching Kids Programming
Lynn is an expert on data technologies. She teaches, consults, and writes on cloud data, business intelligence, and big data solutions. Her latest book is *Smart Business Intelligence Solutions with SQL Server 2008*. Lynn blogs at *www.TeachingKidsProgramming.org*.

Irene Lee
Santa Fe Institute
Irene is the program director of Project GUTS and the principal investigator of GUTS y Girls, NSF-funded programs that engage students in computational thinking and modeling of locally relevant issues as complex systems.

Daniel Mace
Langley Grammar School, UK
Daniel is an Advanced Skills teacher at Langley Grammar School and delivered the Philosophy for Computing lessons for the Digital School House project.

Christopher Michaud
Nebo Elementary School, GA
Christopher currently teaches K–5 Music and CS and coaches the Paulding County First Lego League Robotics Teams. He teaches CS workshops and summer camps with the Georgia Institute of Technology ICE programs.

Karen North
CS Educator, TX
Karen was on the Texas Technology Application Standards for CS writing team. She lobbied to have CS concepts included in K–8 technology essential knowledge and skills. Previously, she taught high school CS, math, and technology systems, and was an elementary school technology specialist.

Kiki Prottsman
University of Oregon
Kiki founded Thinkersmith, a non-profit organization focused on teaching CS as a creative passion. She is enthusiastic about the possibilities for bringing computational thinking to mainstream education.

Michel Resnick
MIT Media Lab
Mitchel is the LEGO Papert Professor of Learning Research and head of the Lifelong Kindergarten group at the MIT Media Lab.

Susan Rodger
Duke University, NC
Susan is a Professor of CS. She leads the JFLAP educational software project (*www.jflap.org*) and works in the area of CS education.

Elisa Rossi
Carroll Middle School, TX
Elisa has taught for 14 years in a variety of computing topics such as keyboarding, personal finance, introduction to business, and now, Media Mania and video game design.

Constance Seiden
Community College of Aurora, CO
Constance is a Faculty member in the Computer Science Department. She has worked to increase young women's interest in technology and CS through a variety of middle school workshops and events in Colorado.

Chris Stephenson
Executive director, CSTA
Chris is a long-time advocate for K–12 CS education. She is the author of several textbooks, white papers, and scholarly articles on CS and adaptive technologies.

Roger Sutcliffe
SAPERE, UK
Roger is an expert in Philosophy for Children (P4C) and past President of the Society for Advancing Philosophical wEnquiry and Reflection in Education (SAPERE).

Elizabeth Sweedyk
Harvey Mudd College, CA
Elizabeth is a Faculty member in the Computer Science Department at Harvey Mudd College and a developer of the *Middle-years Computer Science* (MyCS) curriculum.

Michael Tempel
Logo Foundation
Michael is president of the Logo Foundation, a nonprofit organization he founded with MIT Professor Seymour Papert in 1991. He has been developing Logo software and teaching Logo to students and teachers for more than 30 years.

Aaron VonderHaar
San Diego, CA
Aaron is a Java consultant and volunteer teacher with Wintriss Technical Schools.

Linda Werner
University of California Santa Cruz
Linda is an Adjunct Professor of CS and works with Education, Training, Research (ETR) Associates on researching and developing Alice game programing lessons.

Vic Wintriss
Wintriss Technical Schools
Vic is a Cornell Electrical Engineer and founder of Wintriss Technical Schools, an after-school program in San Diego, CA.

ACM founded CSTA as part of its commitment to K–8 computer science education.

CSTA welcomes your comments.

E-MAIL: *cstapubs@csta.acm.org*

PHONE: 1-608-436-3050

FAX: 1-928-855-4258

csta Computer Science Teachers Association