# Computational Thinking for a Computational World

Digital Promise
Accelerating Innovation in Education

www.digitalpromise.org

# Table of Contents

## Abstract

Computers, smartphones, smart systems, and other technologies are woven into nearly every aspect of our daily lives. As computational technology advances, it is imperative that we educate young people and working adults to thrive in a computational world. In this context, the essential question for American education is: **In a computational world, what is important to know and know how to do?**

This paper argues that computational thinking is both central to computer science and widely applicable throughout education and the workforce. It is a skillset for solving complex problems, a way to learn topics in any discipline, and a necessity for fully participating in a computational world. The paper concludes with recommendations for integrating computational thinking across K-12 curriculum.

# I. Introduction

We live in an increasingly computational world, with computers, smartphones, smart systems, and other technologies woven into nearly every aspect of our daily lives. Computational technology has fundamentally changed how we live, work, and, some would say, even think.[1]

We see these changes all around us: we can now instantaneously communicate with virtually anyone in the world; search and find information on any topic; access or purchase any available product or service; and create content and share it with the world. We can also automate numerous services and activities to improve accuracy and efficiency at a lower cost. In addition, the massive amounts of data we can collect and analyze give us the ability to see the world, and its opportunities and problems, in new ways and on a scale never before possible.

As computational technology advances and becomes even more deeply embedded in our daily lives, it is imperative that we educate young people and working adults to thrive in a computational world. This includes ensuring that everyone is 1) equipped to play a meaningful role in their communities and our democracy; 2) prepared for the changing nature of work; and 3) motivated to pursue lifelong learning so they are empowered to fully participate and proactively shape their futures, and the future of society, work, and technology.

In this context, the essential question for American education is:

> In a computational world, what is important to know and know how to do?

As educators and education stakeholders explore the answer to this question, one thing is clear: the answer is not simply memorizing facts or learning skills that are applicable only to specific tasks, especially when those tasks can be done more quickly and with greater accuracy by technology. Instead, the answer lies in skills that reflect the demands of today's technology-driven world, as well as capabilities that are uniquely human — and will remain so for the foreseeable future.

Because computational technologies are transforming so many dimensions of modern work and life, we concur with others in the education community that computational thinking is a critical part of what is important to know and know how to do in a computational world. Computational thinking is often referenced alongside coding and computer science, but there are important differences between the three:

- **Coding** is the practice of developing a set of instructions that a computer can understand and execute.
- **Computer science** is "the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society."[2]
- **Computational thinking** is "a way of solving problems, designing systems,

> and understanding human behavior that draws on concepts fundamental to computer science… a fundamental skill for everyone, not just computer scientists.”[3]

This paper describes how computational thinking is both central to computer science and widely applicable to other disciplines. It is a skillset for solving complex problems, a way to learn topics across the curriculum, and a necessity for fully participating in a computational world. Because of the importance of computational thinking and the breadth of its relevance, it should be taught across subject areas in K-12 schools.

Following the introduction, Part II examines how computational technologies are changing how we work. Part III introduces coding and computer science education, looking at the evolution and impact of the modern coding and Computer Science for All movements (circa 2013) and the role they are playing in bringing greater awareness to the need for computer science education in K-12 schools. Part IV reports on the current state of K-12 computer science education in the United States. Part V reviews the definition of computational thinking and its relationship to computer science and coding, and the remainder of the paper makes the case for computational thinking as a necessity in a computational world, including recommendations for education stakeholders on how to integrate computational thinking across K-12 curriculum.

# II. Technology is Changing How We Live and Work

In a report titled *The Future of Jobs*,[4] the World Economic Forum (WEF) noted that we are at the beginning of a "fourth industrial revolution" that builds on the digital revolution that began in the middle of the last century. The report described the technological changes that are reshaping economic and daily life, disrupting business practices and social norms.

While acknowledging that the WEF correctly captured the significance of digital technology's effect on how we live and work, some critics of the report contend that labeling this era as the next industrial revolution is incorrect. Jeremy Rifkin, economic and social theorist and author of 20 books on the impact of scientific and technological changes, has instead described[5] what is occurring as a "maturing" of the digital revolution: foundational and other technologies coming together to create "a super-internet to manage, power, and move economic activity across society's value chains."

Regardless of whether we're at the beginning of a fourth industrial revolution or in the middle of a maturing digital revolution, technology is having a dramatic impact on jobs and how we work. For example, smart machines and systems are replacing human labor in some repetitive and predictable jobs such as machining, welding, painting, and assembling parts in certain manufacturing environments. These smart systems are also increasingly being deployed in less predictable jobs, such as operating a vehicle hauling raw materials across a job site and cashiering in a fast-food restaurant. Even some higher skilled tasks such as assisting a customer in opening a bank account or preparing routine legal documents can be done more quickly and with greater accuracy by smart systems.

At the same time, smart systems are also being used to augment and extend human capabilities. An analysis of 900 U.S. occupations showed that technology is now commonly used in all but two job categories.[6] One example of human-centered technology is the use of collaborative robots[7] in warehouses. These small, mobile, relatively inexpensive robots are designed to help human workers perform routine tasks such as finding items in a large or crowded warehouse and transporting items across a warehouse to be packed and shipped. The robots have limited functionality, but what they do helps human workers do their work in less time and frees them up for higher value activities.

Another example is the use of chatbots: programs designed to simulate how a human would behave in conversation. One application of chatbots is conversational ordering: an office manager responsible for ordering office supplies speaks directly to a chatbot on a smartphone or other device; the chatbot verbally confirms the order, and then places it with a selected supplier.

Linked to a commerce application on the backend, the supplier fulfills the order. With conversational ordering, the office manager can order supplies on the fly, without having to stop to fill out and send online forms.

An additional example is the use of intelligent tutoring systems to extend and enhance the impact of expert teachers. These systems simulate one-on-one human tutoring, providing learning activities matched to each student as well as feedback that identifies the exact step of a complex problem-solving process where the student went astray. A teacher can use a tutoring system to provide additional help for struggling students or enable students who work more quickly to move ahead at their own pace.

Computational technology is also transforming numerous high-skill, high-wage professions — especially those that require large amounts of data to be processed and analyzed quickly and synthesized into reports. These include financial services,[8] law, and medicine, where technology brings greater speed, accuracy, and efficiency to routine tasks and creates more capacity for professionals to focus on more complex activities.

## Technologies driving smart systems

In all of these examples, rapid advances in foundational technologies are driving the smart machines and systems that can augment human capabilities: ever more powerful computing and networking systems that both speed up processing and use less power; mobile and remote computing and communications devices and systems that allow people to collaborate across geographical distances; cloud computing for running software and services and storing data on the internet so it can be accessed from anywhere instead of just locally; and machines and devices with built-in wireless networking and sensors.

An example of how these technologies can come together is the **Internet of Things**. In simple terms, the Internet of Things is a network of interconnected smart machines and devices that can collect data and, when connected to automation systems, analyze the data and either act on their own or help people take action. Devices that make up the Internet of Things can be everyday items such as fitness wearables, smart thermostats, and refrigerators. In services and business settings, they can be anything from networked medical devices to environmental monitoring systems.

Take the example of smart refrigerators. Smart refrigerators can sense when items need to be replaced and place orders for delivery of needed items. In addition, refrigerators and other appliances account for about 30 percent of home energy use.

So, if refrigerators and other appliances are thought of "as assets on the electrical grid,"[9] incorporating processing power, networking, and memory would give them the ability to dynamically control their energy consumption. This could reduce utility bills for homeowners and business owners, and enable utilities to meet energy demands with more efficiency.

The evolution of new computing technologies is likely to accelerate as **quantum computing** moves from theory to reality. Quantum computers work differently from classical computers, which are essentially giant calculators that operate based on a set of instructions provided for them by humans. Because classical computers can only work on one thing at a time, the more complex the problem, the more time it takes to solve. Some problems are so complex they cannot be solved by classical computers in a practical period of time. Using a new model of computation that harnesses the power of atoms and molecules, quantum computers have the potential to perform certain calculations at speeds a million times faster than classical computers — putting a whole new class of computational problems now considered intractable within reach.

Once out of the research lab, quantum computing will affect a growing number of technologies that are capable of functions considered more directly and individually "human," and that can replace or augment work activities traditionally done by people. Areas that may see the most significant effects include **artificial intelligence**, **machine learning**, and **robotics**.

**Artificial intelligence** is the ability of a device or software to perform tasks that previously required human intelligence. Examples include visual perception, speech recognition, decision-making, and language translation.

**Machine learning** is an area of artificial intelligence that enables computers to "learn" without explicitly being programmed, similar to the way humans do: by interpreting data and patterns, and self-adjusting based on successes and failures. For example, by analyzing the patterns of a person's choices of music or movies, a smart virtual personal assistant can recommend new options that match their interests based on trends among people with similar profiles.

When embedded in the Internet of Things, artificial intelligence and machine learning can be used, for example, to provide drivers with navigation advice or schedule distributed energy resources such as solar and wind power, based on current usage.

**Robotics** combines mechanical engineering, electrical engineering, and computer science in the design of mechanical systems that can perform tasks and interact with their environment. Sometimes robots can act autonomously, without involving humans; other automated tasks require that robots interact with humans.

The uses of artificial intelligence, machine learning, and robotics are vast, and include:

- Intelligent agents that combine speech recognition, knowledge of and the ability to interact with users in varied ways, including natural and conversational language, with connections to data sources and web or mobile apps. These capabilities combine to enable them to perform useful tasks, such as those in the conversational ordering application described earlier.

- Intelligent systems that can find patterns in large volumes of all types of data to find answers to questions or draw informed conclusions, and even "learn" complex problem-strategies. Examples of such systems are IBM's Watson[10] and DeepMind's AlphaGo.[11]

- Autonomous or self-driving vehicles that are equipped with technology capable of sensing environmental information that allows them to perform safely without a human operator. Examples of such vehicles currently in use include driverless trucks hauling materials across a job site and self-driving tractors and combines in agriculture. Already, some passenger cars can operate mostly without human intervention, matching driving speeds to surrounding traffic, keeping within a lane or automatically changing lanes, self-parking, and being summoned from a parked position. Someday, we can imagine that public roads will be populated with millions of driverless trucks, taxis, and passenger cars.

- Robots and robotic systems that can sense and take action in the physical world, either self-guided or guided by a human operator. One example is the use of a robot diver[12] that can dive to environments and depths that are too dangerous or beyond the physical limits of human divers. The human operator remains on land or a ship while visual and haptic feedback systems let the operator "see" everything the robot sees and "feel" everything the robot touches. The robot can gather materials and use tools, which makes it valuable for salvage operations or underwater construction.

The implications of these technologies, which can replace, extend, or augment human capabilities, are profound: they affect the workplace and society at large.
For example, consider how the language that is commonly used to describe these technologies — smart devices, machine learning, artificial intelligence — blurs the boundary between human and nonhuman characteristics. With these implications in mind, we return to the essential question posed by this paper: in a computational world, what is essential for people to know and know how to do?

This essential question has breathed new life into efforts that have been underway for decades to bring computing into K-12 education.

# III. The Coding Movement and Computer Science for All

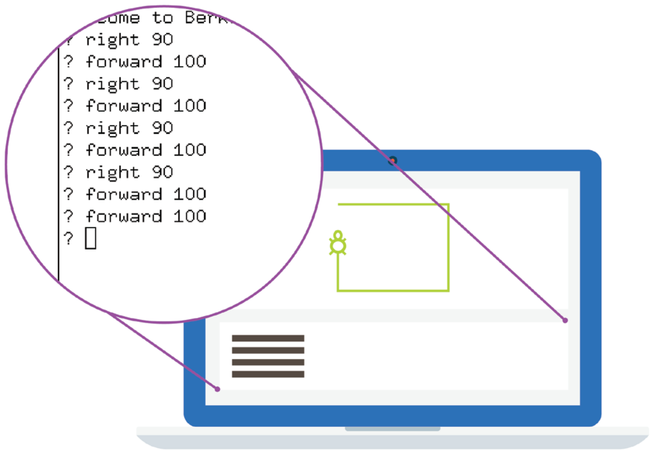
In recent years, leaders from all walks of American life have joined a movement that advocates for every student to learn how to code.[13] Parents are on board: a 2015 Google/Gallup poll found that 9 out of 10 U.S. parents want their kids to learn how to code at school, with parents in lower income groups placing an even higher value on learning coding than those with higher incomes.[14]

As a result of this widespread enthusiasm, millions of dollars of private and public funds have been contributed to coding programs in our K-12 schools as well as to organizations in the movement that design and deliver programs *(see Appendix A: Leading K-12 Coding Non-Profits).* With this level of attention and investment, it is worth asking: what is the value of teaching children to code?

## Coding for kids is not a new idea

The idea of helping children to learn computation is not new. Seymour Papert, mathematician, computer scientist, educator, and lifelong advocate for equity and inclusion, argued as early as 1968 that computing could be a vehicle for learning. With colleagues, he developed a theory of learning called "constructionism" (building on Jean Piaget's theory of constructivism), which posited that people learn most effectively when they are actively engaged in constructing things in the world. Papert argued that learning how to program computers could provide a meaningful context for children to learn how to think about themselves, their learning, ideas, and experiences. He also argued that computers could be powerful tools to help children make sense of the world.

In 1967, Papert, who co-founded the MIT Artificial Intelligence Lab with Marvin Minsky, partnered with Wally Feurzig, Cynthia Solomon, and others to develop a programming language for children that would enable them to explore powerful ideas in a variety of domains and engage intellectually in challenging content. They called the language "Logo." The most widely recognized feature of Logo was the use of a robotic turtle that sat on the floor; children could move it by writing Logo programs on an attached teletype. The Logo team later designed an on-screen version of the turtle that could be used to draw shapes, designs, and pictures, and create animations and games on a computer screen.



```
 come to Berk
? right 90
? forward 100
? right 90
? forward 100
? right 90
? forward 100
? right 90
? forward 100
? forward 100
? 
```

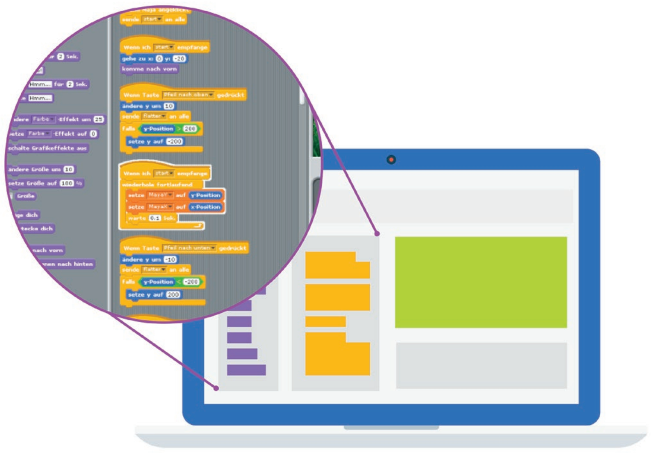
The ability to program a turtle to move about in space provided a context for mathematical ideas to have obvious power and utility. Logo also helped make mathematical abstractions concrete by providing immediate feedback — not by telling students they were right or wrong, but by inspiring students to "debug" (identify and correct errors) problems they observed in the mathematics expressed in their own programs.

In 1971, Papert and Solomon wrote a paper, "Twenty Things to Do with a Computer,"[15] that provided examples of how computers could be used to take action, not just solve mathematical problems. The paper could be considered a blueprint for today's maker movement in education.

In the late 1970s, with the advent of personal computers, Logo emerged from MIT and a handful of other research labs and began to make its way into K-12 classrooms. The language gained momentum in the 1980s with the publication of Papert's book, *Mindstorms: Children, Computers, and Powerful Ideas*.[16] The book provided examples of how Logo could be used to engage students in their own learning, which sparked teacher interest in bringing Logo into their classrooms.

In the mid-1980s, while at the MIT Media lab, Mitchel Resnick and Steve Ocko developed a system that interfaced Logo with motors, lights, and sensors built into machines made of LEGO bricks. The collaboration led to a popular line of educational and consumer kits that enabled students to build customizable, programmable robots. Called LEGO MINDSTORMS after Papert's book, the kits have inspired millions of young learners all over the world to learn programming and build robots. In addition, throughout the 1980s, dozens of books were published on how to teach Logo programming, conferences were convened, and research was conducted.

A new visual programming language inspired by Logo, Scratch,[17] was developed in 2004 as a project of Resnick's Lifelong Kindergarten Group[18] at the MIT Media Lab. Designed to help students "think creatively, reason logically, and work collaboratively," Scratch is provided free of charge to schools, educators, and parents, and has been used by millions of students to create and share interactive media. Users have shared nearly 25 million Scratch projects in an online creative learning community hosted by MIT.



Despite the popularity of Scratch and other tools, and the decades that have passed since Papert began his work, the vision of all students programming computers in order to learn about the world has not come to fruition. Nevertheless, it is important to frame today's efforts in the context of this historical lineage so that researchers and practitioners can learn important lessons from past successes and failures.

## Coding catches on

The modern coding movement ignited[19] in 2013, when the nonprofit Code.org introduced its Hour of Code,[20] a one-hour introduction to computer science for K-12 students. The work of other longtime proponents of teaching K-12 students to code — including the Association for Computing Machinery (ACM), the Computer Science Teachers Associations (CSTA), the International Society of Technology in Education (ISTE), and the National Science Foundation (NSF) — had not previously garnered such widespread popular support. With effective marketing, a simple point of access, and a narrative that emphasized workforce development, Hour of Code triggered a social movement that engaged a broad group of stakeholders: computer science educators and researchers, education leaders, curriculum developers, classroom teachers, nonprofits, policymakers at both the federal and state levels, and employers across industries.

A compelling argument in support of everyone learning how to code was that job opportunities for programmers in the United States had expanded beyond technology into all industries. A 2015 report from Burning Glass[21] confirmed this idea, finding that half of job openings for programmers were in industries outside of technology, most notably in finance, manufacturing, and healthcare.

Another argument for learning to code was that in a world economy increasingly defined by computational technology, coding skills will be essential to many jobs in the future, and not just for programmers. The Burning Glass survey found that in 2015, 7 million U.S. jobs representing 20 percent of "career track" jobs were in occupations that value coding skills. These included Information Technology (IT) workers, data analysts, artists and designers, engineers, and scientists.

To many in the education community, the movement's focus on workforce development was troubling because it could be interpreted to mean that the main purpose of education is to help students get a job. While career readiness is a legitimate expectation of education, so is preparing citizens to participate in our democracy, increasing social cohesion by creating common experiences, and, in the words of John Dewey, "to nurture individuals to discover their full power and potential" and use what they've learned for the greater good.[22]

Another concern of educators — which had been shared by organizations championing expanding access to computer science in previous decades — was that the main beneficiaries of the new movement seemed to mirror the current profile of the professional programmer: white, middle- and upper-class, mostly male. However, in a blog post[23] in the *Washington Post* in 2016, education researchers Jane Margolis and Yasmin Kafai offered the view that the movement had the potential to increase the participation of people who have historically been underrepresented in computer science and other STEM fields. They wrote, "Computer science can help interrupt the cycle of inequality that has determined who has access to this type of high-status knowledge in our schools."

Also building on the efforts of organizations that had been working to expand access in previous decades, Margolis and Kafai painted a broader vision for the value of knowing more than just how to program or use computers: “Being a digital native today isn’t just about browsing the web, using technology to communicate, or participating in gaming networks. It really involves knowing how things are made, breaking down and solving problems, designing systems, contributing through making, and understanding social and ethical ramifications. We see how computers in any form and place have become an inextricable part of our social lives—not just how we interact but also how we contribute.”

Despite the emphasis on “coding,” as reflected in the names of many of the organizations comprising the popular movement, most also emphasized and advocated for the broader academic discipline of computer science. However, even though the terms “coding” and “computer science” are sometimes used interchangeably, they are not synonymous. Coding involves writing a set of instructions a computer can execute. Computer science is the study of computers and how computation can be organized and analyzed, including social and ethical questions that surround the use of computers.

Equally important, the movement put equity front and center as a guiding principle. For example, Code.org describes its mission[24] as “expanding access to computer science” and “increasing participation by women and underrepresented minorities.” Nonprofits such as Black Girls Code,[25] Girls Who Code,[26] and CodeNow[27] were founded specifically to help students who have not traditionally had access to these kinds of learning opportunities.

## IV. The Current State of Computer Science Education in K-12 Schools

Because education in America is highly decentralized, there is no single authority at the national level guiding the purpose and implementation of opportunities to learn about computing. This means that even if access to opportunities to learn were expanded to every student in the country, the goals and implementation models for computer science curriculum could vary across states and districts, meaning that not all students would have the same learning experiences or outcomes.

Amid this diversity, there are common motivations emerging for K-12 districts and schools offering computer science education. Here are four:

**1. Developing job skills.** K-12 education leaders recognize that there is a gap between graduating students' skills and employers' needs and values. Next Generation Science Standards,[28] for example, are designed to address what the "modern day scientist" is actually doing, and help build relevant skills and competencies early. More than 40 states have shown interest in the standards and 18 states plus the District of Columbia have adopted them, representing 35 percent of the nation's students.

Computer science skills are increasingly needed in a broad range of professional settings, even in non-technology fields.[21] This skills gap is likely to worsen with the technological changes described earlier in this paper, which will dramatically alter the nature of employment and skills in demand.[29]

**2. Shifting pedagogical practices.** Some educators and leaders promoting computer science classes see computer science as an opportunity to deepen learning by employing the principles of constructionism,[30] which is Papert's theory of learning-by-making described earlier in this paper. Advocates of constructionism have long supported the idea of students using computers and programming languages to learn by creating digital artifacts, including applications, digitally controlled physical objects, and digital stories that are told on many platforms and across multiple media. Consequently, some educators see computing education as the best opportunity to apply constructionist principles to curriculum and pedagogy.

**3. Advancing a broader set of higher order skills.** Schools and districts are deeply engaged in moving beyond traditional content knowledge to build a broader range of higher order skills, including creativity, collaboration, critical thinking, and communication. Notably, new standards such as the Common Core State Standards[31] and Next Generation Science Standards integrate computational skills in activities such as analyzing data, creating simulations, and modeling. Even though state adoption of these standards is not universal and implementation varies, the standards influence many state policy discussions. Schools see computer science as an opportunity to build higher order skills while targeting some of these standards.

**4. Broadening participation in computer science education and careers.** Even though the participation of women in computer science has seen gains in recent years, it remains disproportionately low. For example, even though women received 55 percent of U.S. bachelor's degrees granted in 2015,[32] women accounted for just 16 percent of the computer science bachelor's degrees reported.[33] The statistics on underrepresented minorities tell a similar story: in 2015, underrepresented minorities accounted for just 11 percent of computer science bachelor's degrees reported.[33]

As with all achievement gaps, the diversity gap begins with socioeconomic disparities: children do not come to school with equal advantages. It is difficult for schools to change achievement gaps, and thus it should be no surprise that the diversity gap in computer science in college and careers continues throughout K-12 schools. Although it may be tempting to blame this on the schools

Instructional approaches can embrace one or more of these motivations for teaching computer science. Consider how the following classroom learning activities can help to develop job skills, shift pedagogical practices, advance higher order core competencies, and create a more relevant and inclusive learning environment for students:

- Creating websites and mobile apps for authentic users, including:
  - Advocacy campaigns for social causes
  - Websites for local small businesses and nonprofits
- Programming computer games, such as:
  - Educational games for younger students
  - Interactive stories about social issues
- Exploring newly available technologies used in industry, such as:
  - Virtual reality and augmented reality
  - Parametric 3D modeling and digital fabrication
- Analyzing and communicating with data, including:
  - Visualizing the nutritional values of foods in the school cafeteria
  - Analyzing the performance of favorite athletes and teams
- Creating computational models to simulate and study complex systems, such as:
  - Investigating the potential impact of seismic activity on a neighborhood
  - Visualizing and reconfiguring local traffic patterns
- Tackling real-world challenges, such as:
  - Collecting and analyzing local weather and climate data and making recommendations for local action to combat climate change
  - Developing potential solutions to the United Nations Sustainable Development Goals

themselves, it is also quite difficult for schools to assemble qualified instructors for a new course, especially with a red hot job market for the same skills in industry. Not surprisingly, schools in poor neighborhoods have an even harder time finding highly qualified instructors and offering courses. A 2016 report[34] exploring the gap in secondary schools found that underrepresented groups face both structural and social barriers that create disparities in learning opportunities. For example:

- Black students are less likely than white students to have computer science classes in the schools they attend: 47 percent versus 58 percent, respectively.
- Black and Hispanic students are less likely than white students to use a computer at home: 58 percent, 50 percent, and 68 percent, respectively.
- Male students are more likely than female students to be told by a parent or teacher that they would be good at computer science: 46 percent versus 27 percent being told by a parent; 39 percent versus 26 percent being told by a teacher.

The demographics of U.S. high school students taking Advanced Placement exams offers another barometer of diversity in various fields of study. In 2016, more than 50,000 U.S. high school students took the Advanced Placement (AP) Computer Science Level A exam — more than double the number in 2012, a 17.3 percent increase over the previous year, and a new record.[35] However, the gender and racial makeup of the students who took the exam is very different from the demographics of the total student population. For example, in 2016, girls made up just 23 percent of the students who took the exam nationally, and underrepresented minorities just 16 percent.

Diverse challenges need diverse teams to address them — anything less holds everyone back.

In the 2016–2017 school year, the College Board introduced a new computer science exam, AP Computer Science Principles. Data on the administration of the first test[36] show that more than 29,000 female students took the exam, bringing the percentage of girls taking one of the AP computer science exams to 27 percent. Results for underrepresented minorities showed similar gains, showing that schools are making progress on these challenging issues. In the spring of 2017, more than 22,000 underrepresented minorities took the AP Computer Science Principles exam, bringing their percentage participation in one of the AP computer science exams to 20 percent. While this is progress, participation rates for female students and underrepresented minorities is far from representative.

In addition to issues of equity and educational opportunity, there are at least three economic reasons why everyone should act to ensure gender and racial diversity in computer science and STEM education and careers more broadly. First, STEM careers tend to pay well, and women and underrepresented minorities who enter STEM fields or get jobs that require STEM skills will have more earning power than they would have otherwise. Second, research shows that when women prosper, their families and the communities in which they live benefit[37] through more economic opportunity and growth. Third, there is growing recognition that diversity is essential to excellence[38] in many areas of endeavor, from problem solving to product development. As Karen Phillips described in Scientific American, "This is how diversity works: by promoting hard work and creativity; by encouraging the consideration of alternatives even before any interpersonal interaction takes place." Diverse challenges need diverse teams to address them — anything less holds everyone back.

## Computer science education policies vary from state to state

Given the decentralized nature of the U.S. education system, it is not surprising that computer science education policies vary in states across the country. Code.org recommends that states pursue nine policy ideas to embed computer science in K-12 education.[39] These recommended policy ideas cover crucial areas such as funding, standards, certification pathways for teachers, graduation credit, and implementation guidelines.

States adopt different policies on their own timelines, making the policy narrative for computer science education in the United States complex and constantly shifting. A brief summary of state policies as of August 2017[40],[41] is as follows:

- Ten states have adopted computer science education standards.
- Twenty-nine states have adopted certification pathways for teachers to teach computer science.
- Two states have ratified statewide plans for computer science education.
- Thirty-five states allow computer science to count as a mathematics or science credit for the purposes of high school graduation.
- Some schools are integrating computer science into existing career and technical education (CTE) programs, expanding the reach of computer science beyond the academic program.

## Progress is being made

Since their original publication in 2004, the CSTA K-12 Computer Science Standards[42] have been in use by states developing curriculum standards for computer science.

In 2016, the ACM, the Computer Science Teachers Association, Code.org, and more than 100 experts from the computing and education communities published a K-12 Computer Science Framework,[43] offering conceptual guidelines for computer science education more broadly.

In terms of tools and resources, the CSforAll Consortium[44] is a popular resource hub for programs and teachers in classrooms across the country. Inspired by President Obama's[45] Computer Science for All initiative launched in 2016, the CSforAll Consortium is led by a team at CSNYC,[46] an organization committed to computer science for all New York City students, and with initial funding from the National Science Foundation. Today, the Consortium brings together more than 300 researchers, content providers, education associations, and funders to help K-12 schools deliver inclusive, rigorous, and sustainable computer science education.

At the high school level, the Computer Science Advanced Placement exams are more widely taught and standardized than any other computer science education course. Still, Advanced Placement courses are not universally taught in high schools and, when they are, students are most often required to have a high level of achievement to enroll in them. This limits their potential to expand opportunities to all students.

In September 2017, President Trump called on the U.S. Department of Education to direct $200 million a year to STEM fields, including computer science.

Looking at the computer science education field as a whole, there is a clear priority among advocates to increase adoption of computer science in schools and reach more students. By this measure, the movement has made progress: in 2016, 40 percent of secondary schools offered at least one computer science class, up from 25 percent the year before.[47]

In addition, some of the country's largest school districts, including New York City, Los Angeles, Chicago, and San Francisco, have committed to making computer science education available to all students.

## Barriers to success remain

Adoption of computer science in education has been growing rapidly, if unevenly, yet barriers to success remain. Here are three:

**1. A severe shortage of classroom teachers with the skills necessary to teach computer science.** One of the most critical pieces of computer science education policy is training for pre- and in-service teachers. In all the states with teaching certifications in computer science, only 75 college graduates received certification in 2016.[48] Pre-service training is not yet producing enough teachers certified to teach computer science to address even a tiny fraction of the demand. Rather than waiting for the pipeline of certified pre-service teachers to grow, experts suggest that states pursue a phased strategy to equip in-service teachers to teach the subject. By integrating professional development for primary school teachers and offering "endorsements" for qualified secondary school teachers, states can more quickly spread computer science learning opportunities.

**2. A lack of bandwidth or knowledge to look ahead to how computer science education should evolve.** When we asked leaders in the computer science education movement how they think computer science education should evolve, the answer was that schools are so focused on increasing participation in computer science now that they have little time to think about how computer science might evolve in the future and what that might mean for their initiatives. Nevertheless, K-12 districts and schools must focus on continuous improvement and iteration in computer science initiatives to better serve an increasingly diverse student population and adapt to a changing world. If the access and participation goals of the Computer Science for All movement are to be met, it will be necessary to change how it is taught based on research about what works, what does not, and for whom. Given the limited capacity of K-12 schools to do this on their own, advances are more likely through partnerships among K-12 educators, education researchers, and computer science professionals.

Inclusive pedagogies need to be developed and broadly implemented in computer science courses to further encourage diverse and equitable participation.

**3. Equitable participation continues to be a challenge**. The persistent lack of diversity in computer science courses leads some to question whether expanding computer science education in its current form will lead to the greater participation we seek. One action advocated by some is to make a computer science course a requirement for high school graduation. This would require that all students have access and that girls and underrepresented minorities in computing and other STEM fields have the same opportunity to learn and master the skills. An alternative would be to allow computer science courses to count toward high school graduation requirements in math or science. This action would likely serve to expand opportunities for all students, not just for those who have space in their schedules for elective courses. For maximum impact with any of these approaches, inclusive pedagogies need to be developed and broadly implemented in computer science courses to further encourage diverse and equitable participation.

# V. The Central Role of Computational Thinking

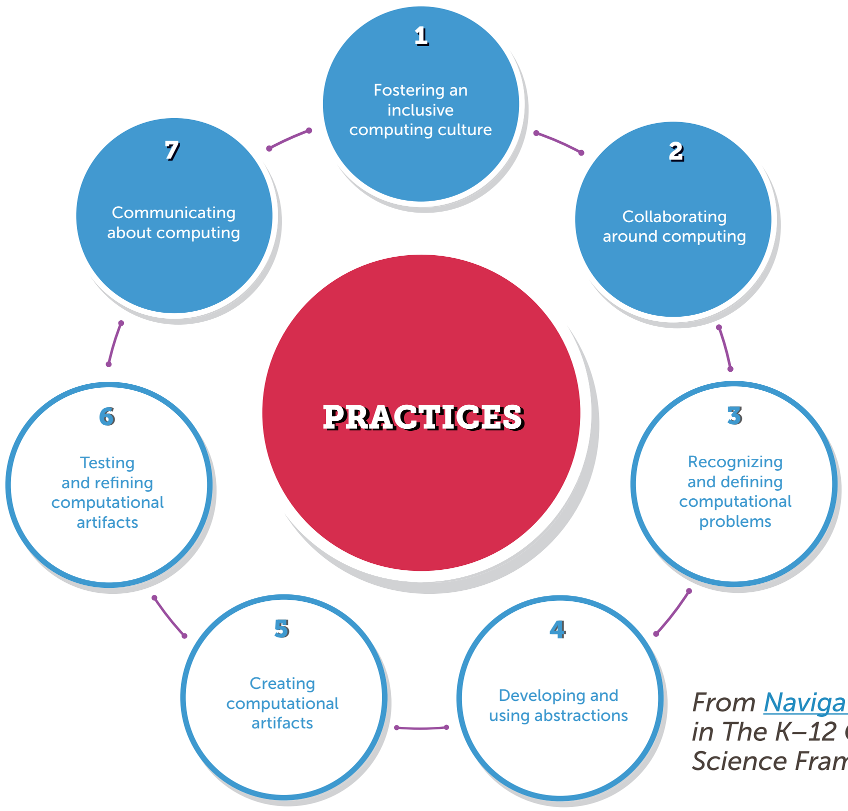
Computational thinking is recognized as core to computer science, and its relevance for learning extends beyond that academic discipline.

The K-12 Computer Science Framework referenced in the previous section of this paper identifies seven core practices[49] of computer science. They are:

1. Fostering an inclusive computing culture
2. Collaborating around computing
3. Recognizing and defining computational problems
4. Developing and using abstractions
5. Creating computational artifacts
6. Testing and refining computational artifacts
7. Communicating about computing

The framework places computational thinking at the heart of the seven core practices, identifying practices 3 through 6 as components of computational thinking and practices 1, 2 and 7 as complementary.

The relationship between programming/coding, computer science, and computational thinking makes sense: for computers to help people solve problems, they must be given instructions about what to do in a language they can understand. The skill required to tell a computer what to do is programming.


PRACTICES
1
Fostering an inclusive computing culture
2
Collaborating around computing
3
Recognizing and defining computational problems
4
Developing and using abstractions
5
Creating computational artifacts
6
Testing and refining computational artifacts
7
Communicating about computing

*From Navigating the Practices in The K–12 Computer Science Framework*

The thought process behind programming is computational thinking. Both computational thinking and programming are necessary in the study of computer science.

However, determining the purpose, structure, and desired outcomes of an application is often more demanding than the actual coding. When viewed this way, computational thinking both enables and transcends programming—a view introduced by Jeanette Wing in her seminal article, "Computational Thinking," published in the March 2006 issue of the *Communications of the ACM*.[3] Wing defined computational thinking as "a way of solving problems, designing systems, and understanding human behavior that draws on concepts fundamental to computer science."

Wing's statement opens the door to computational thinking as useful in many fields — an idea Wing explicitly proposed when she wrote, "Computational thinking is a fundamental skill for everyone, not just for computer scientists."

In this broader context, computational thinking skills include:

- Gathering and organizing data to investigate questions and communicate findings
- Expressing procedures as algorithms (that is, a series of logical, precise, repeatable steps that delivers an expected result) to reliably create and analyze processes
- Creating computational models that use data and algorithms to simulate complex systems
- Using and comparing computational models to develop new insights about a subject

The skill required to tell a computer what to do is programming. The thought process behind programming is computational thinking.

These practices of computational thinking benefit both cutting-edge research and everyday life. For example, when a hurricane is approaching, a meteorologist on TV may use a computational model to demonstrate the various paths that the storm may take as any number of interdependent variables change. An astrophysicist may similarly use computational thinking practices to develop simulations and new theories about the collisions of black holes.

Any number of problems can be reframed and partially or wholly solved through computational solutions. Examples include optimizing workflow in a restaurant kitchen to provide timely service and order accuracy; increasing the speed and accuracy of processing a loan application; reducing call wait times in a customer service center; easing traffic bottlenecks in a busy metropolitan area; predicting the spread of a disease in a geographic area; and even figuring out how to build floating piers[50] spanning a lake that could allow more than a million visitors to view the landscape from never-before-available vantage points (as artists Christo and Jeanne-Claude did in 2016).

Although some of these scenarios may be more familiar than others, the underlying practices of computational thinking are the same. When faced with a problem, a computational thinker reframes the problem so it can be represented by a model of data and algorithms. Not satisfied with just any solution, a computational thinker considers and tests many possible computational models before selecting one to implement — or decides there is no effective computational solution to the problem. Since computational models are representational, a computational thinker critically analyzes which representation yields the most clear, efficient, and accurate approach, and questions whether there are other ways to understand the problem or improve the model.

It is important to note, however, that certain kinds of problems cannot be solved through computational solutions, and likewise that computational thinking is not the only approach to understanding and solving problems. A 2017 article[51] in American Scientist noted that computational solutions are by themselves inadequate for problems that require social cooperation to resolve. Issues known as "wicked problems," moreover, have no clear problem framing because of different social interpretations of the causes, and may have interdependencies such that any solution could cause or worsen other problems. For example, there are many possible ways to explain the sources of poverty, and any proposed solution has the potential to cause unforeseen consequences in the short and long term. For socially-embedded and wicked problems like these, computational thinking is only one of a number of strategies that could be employed to define the problem and develop possible solutions.

When defining what computational thinking is, it is also helpful to establish what it is not: that is, humans thinking like a computer. In fact, it is just the opposite. Computational thinking is a uniquely human ability.

As Wing noted, "Computational methods and models give us the courage to solve problems and design systems that no one of us would be capable of tackling alone. Computational thinking confronts the riddle of machine intelligence: What can humans do better than computers? What can computers do better than humans? Most fundamentally, it addresses the question: What is computable?"[3]

When defining what computational thinking is, it is also helpful to establish what it is not: that is, humans thinking like a computer. In fact, it is just the opposite. Computational thinking is a uniquely human ability.

In his 2007 article, "Computational Thinking is Pervasive," published in the *Journal of Scientific and Practical Computing,* Alan Bundy corroborated Wing's view of computational thinking as broadly applicable, noting that computational thinking was "influencing research in nearly all disciplines, including sciences and humanities." Bundy also asserted that at a more fundamental level it was *changing how we think*: "Computational concepts provide a new language for describing hypotheses and theories. Computers provide an extension to our cognitive faculties. If you want to understand the 21st century then you must first understand computation."[52]

Viewed this way, computational thinking can be characterized in much the same way computer programming was by Papert 50 years ago: that is, computational thinking is both a skill to learn and a way to learn — to create, discover, and make sense of the world, often with computers as extensions and reflections of our minds. Consider a few examples that span different domains:

- **Using computational models and simulations, we can better understand large, complex systems with many interrelated parts.** Examples of questions that can be answered include: how does a market influence the prices of the goods that comprise it? What happens to an ecosystem when one species is removed? Are there ways to better predict the effects of climate change?

- **By creating algorithms, we learn to develop processes with logical, precise, repeatable steps.** For example, how might we document and compare the effectiveness of different strategies for solving the same problem? What might students learn about managing complexity by formalizing routines into algorithms?
- **Learning how to work with data supports inquiry into many types of phenomena.** For example, what can parish records of births, marriages, and deaths from the 1500s tell us about early modern English history? What might a political science class learn by examining the patterns that emerge when juxtaposing data on droughts and drone strikes?

## How it all fits together

Given how ubiquitous computing is in our daily lives, there is a push in the education community and beyond to declare particular skills or competencies associated with computation a "literacy." What differentiates a skill from a literacy is that a skill is specialized (a specific thing an individual knows or knows how to do), while a literacy is generalizable (enabling an individual to form coherent interpretations in any situation and act effectively). Framed this way, reading and writing are skills; textual literacy[53] occurs when an individual uses reading and writing effectively for personal and social purposes.

In our increasingly computational world, a number of skills related to computers have been promoted as new literacies, including ICT (information, communications and technology) literacy, digital literacy, media literacy, information literacy, computational participation,[54] and computational literacy,[55] to name a few. The nuances that distinguish these terms can be subtle, and they are an ongoing topic of discussion among education researchers.

In her article,[56] "Understanding Computer Programming as a Literacy," published in *Literacy in Composition Studies* in 2013, and in a subsequent book, *Coding Literacy: How Computer Programming is Changing Writing*,[57] Annette Yee explores the idea of programming as a literacy. However, when it comes to labels, Yee substitutes the broader term "computational literacy" for programming, because "it helps us better understand the social, technical, and cultural dynamics of programming... [and] also enriches our vision of 21st century composition."

Education researchers Shuchi Grover and Roy Pea concluded in their article,[58] "Computational Thinking in K-12: A Review of the State of the Field," published in 2013 in *Educational Researcher*, that while the terms computational literacy and computational thinking are often used interchangeably, the term computational thinking seems preferred in research and practice.

In the end, it is not the purpose of this paper to define and discuss all of these terms.

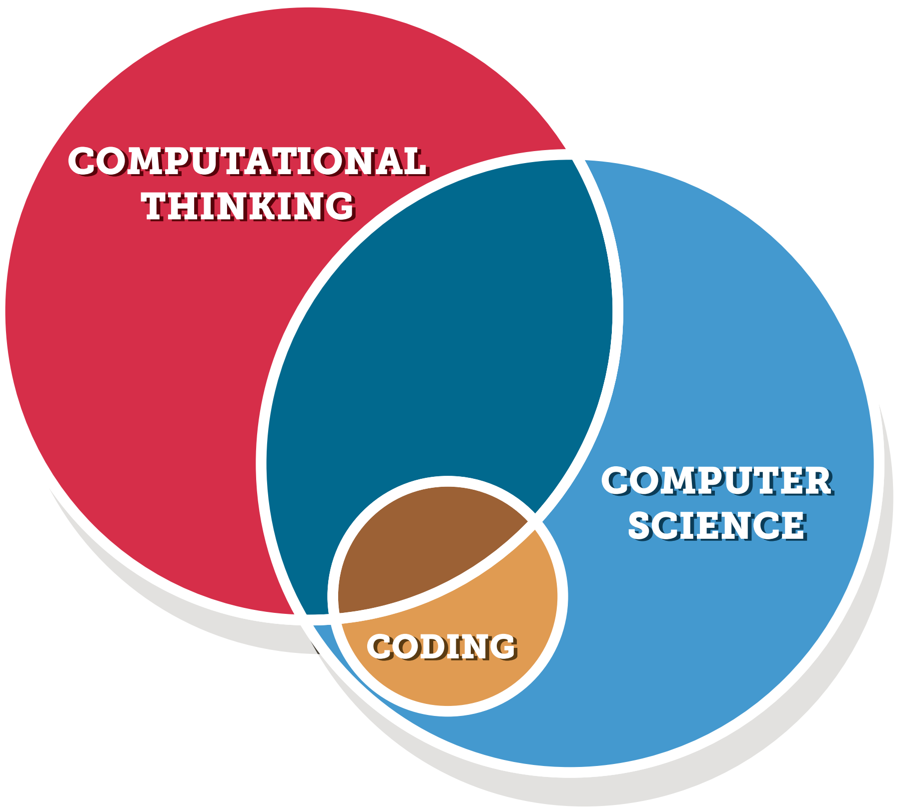
Computational thinking is both a skill to learn and a way to learn — to create, discover, and make sense of the world, often with computers as extensions and reflections of our minds.

Instead we can review the definitions that have been presented in this paper and how they relate to one another. To summarize:

- **Coding** is the practice of developing a set of instructions that a computer can understand and execute.
- **Computer science** is "the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society."[2]
- **Computational thinking** is "a way of solving problems, designing systems, and understanding human behavior that draws on concepts fundamental to computer science… a fundamental skill for everyone, not just computer scientists."[3]

Defined this way, coding can be considered a technical skill; computer science is an academic discipline; and computational thinking is a problem-solving process central to computer science that can be applied more broadly to problem solving and learning in any discipline.

## The Relationship between Coding, Computer Science, and Computational Thinking



Each of the three competencies explored in this paper is valuable in its own right and simultaneously linked to the other two. Just as coding and computer science have been recognized for their importance, computational thinking should be recognized as critical for participation in today's computational world.

# VI. Integrating Computational Thinking Across K-12 Curriculum

Given the importance of computational thinking, how can K-12 education evolve to include computational thinking for all?

There is good news: computational thinking is already underway in schools. A Reach Capital Field Report on K-12 Computer Science,[59] published in August 2017, showed that some components of computational thinking are already part of widely adopted standards in K-12 education. In addition, as described earlier in this paper, computational thinking is already recognized as core to computer science. As computer science enters more schools, there is potential for computational thinking to scale in those contexts.

However, progress is still needed to fully integrate computational thinking practices in schools. Much of the current literature on computer science education fails to recognize the distinctions between computer science and computational thinking modeled in the diagram above. In addition, our own research[60] indicates that while computational thinking is recognized as a core competency of computer science, it is infrequently acknowledged as a core competency applicable across all disciplines. And even where existing standards include aspects of computational thinking, there is room for improvement: they do not yet encompass the full scope of computational thinking practices, and could better connect to the technologies and media that are increasingly embedded in students' lives. To move toward full participation by all students in a computational world, computational thinking practices should be common tools for learning across disciplines.

Advocating for computational thinking in K-12 curriculum does not replace or compete with efforts to expand computer science education: on the contrary, it complements them.

It is important to note that advocating for computational thinking in K-12 curriculum does not replace or compete with efforts to expand computer science education: on the contrary, it complements them. Where computer science is not yet offered, integrating computational thinking into existing disciplines can empower educators and students to better understand and participate in a computational world. And schools already teaching coding and computer science will benefit from weaving computational thinking across disciplines in order to enrich and amplify lessons that are beyond the reaches of computer science classes.

## Challenges on the road ahead

Significant questions remain unanswered when it comes to integrating computational thinking for all. For example, the research community has identified issues, including the need for an agreed-upon definition for computational thinking; whether coding, computer science, and computational thinking can legitimately be separated; the best and most inclusive pedagogy for promoting computational thinking in children; how computational thinking can be assessed; and whether it is good for everyone.[60, 61] This paper seeks to surface these questions, and encourage further collaboration between practitioners and researchers to fully answer them.

Further, if schools, districts, and states are struggling to find qualified computer science teachers, how will they find teachers qualified to deeply integrate computational thinking across all K-12 curricula? The challenge is one of market dynamics: how to match growing demand for computational skills with an increasing supply of prepared teachers in all academic disciplines.

While the above barriers to computational thinking in K-12 curriculum must be overcome to achieve computational thinking for all, an important initial consideration for schools seeking to integrate it deeply across all disciplines is to determine what are they seeking to accomplish.

For example, if the goal is to develop immediate job skills, that might lead schools to offer coursework in programming languages that are currently in demand by employers. Focusing on this goal requires attention to programming languages that may become obsolete, and an emphasis only on coding means that other important principles of computational thinking may be ignored.

If the goal is to achieve full participation in a computational world, then schools might pursue a strategy to introduce computational thinking across disciplines, in addition to any computer science courses that may already exist. Beyond introducing a new discipline, this implies a cultural shift in which all educators value, understand, and use the practices of computational thinking in their teaching. If successful, the skills students develop may be richly valuable over a lifetime.

To address the teacher shortage challenge, Digital Promise has developed 10 educator micro-credentials that focus on the key concepts and pedagogical practices that support the development of computational thinking. Micro-credentials recognize educators for the skills they learn throughout their careers in a way that is competency-based, on-demand, and personalized.

These 10 micro-credentials identify methods educators can use to help students build the core computational skills described earlier in this paper. They also recognize key pedagogical practices for teaching computational thinking, such as creating inclusive learning environments, integrating computational thinking into existing curriculum, and assessing computational thinking (see Appendix B: Digital Promise Computational Thinking Educator Micro-Credentials). Expertise in computer science and coding is not a prerequisite for earning these micro-credentials, and teachers across disciplines can develop these skills for teaching computational thinking.

The computational thinking micro-credentials benefit everyone: educators are better supported in charting their own pathways of professional growth; schools are better able to meet new challenges of education in a computational world; and students have a better chance of engaging in computational thinking across a broad range of disciplines.

# VII. Recommendations

In this section we offer recommendations for educators and education stakeholders who support integrating computational thinking into K-12 education.

## Awareness and Advocacy

- Encourage corporate, nonprofit, and government advocates to hold competitions, showcases, and celebrations that promote the value of computational thinking in different domains of education and the workforce.
- Engage a broad diversity of educators across the curriculum in computational thinking by developing partnerships with professional organizations dedicated to teaching disciplines, including social studies, literature, biology, physics, mathematics, visual and performing arts, and physical education.
- Leverage the overlap of active learning approaches (including maker learning, design thinking, challenge-based learning, and others) to create a more robust network of advocacy for pedagogies supportive of computational thinking.
- Launch social and other media campaigns to build understanding and support for computational thinking among parents, community organizations, and policymakers.

## Curriculum and Resource Development

- Build on existing resources to develop a library of Open Education Resources (OER) that aligns with widely adopted curriculum standards (e.g., Common Core State Standards, Next Generation Science Standards) and can be continuously improved.
- Develop specific approaches and curricular materials that advance inclusiveness and promote diversity.
- Develop curriculum and resources on computational thinking through the lens of social studies, history, and civics to teach the limitations and impacts of computing.
- Develop resources to support and leverage networks of practice for out-of-school programs that engage youth in computational thinking.

## Teacher Training

- Expand innovative pathways for professional learning, including additional micro-credentials, to support educators of different age groups and content areas.
- Develop and scale opportunities for professional development for in-service teachers, both online and in person.
- Create resources to support school administrators who are integrating computational thinking into their schools.

## Research

- Conduct research to determine the most effective ways to incorporate computational thinking across all subject areas at scale.
- Use design-based implementation research methods utilizing researcher-practitioner partnerships to support, document, and scale effective, inclusive and engaging pedagogical and assessment strategies, with a particular focus on women and underrepresented minorities.
- Develop implementation-ready strategies for teachers and school leaders to make computational thinking more inclusive, equitable, and useful for the full diversity of learners (e.g., culturally diverse, neurologically diverse, etc.).

## VIII. Conclusion

Returning to the essential question posed in this paper: in a computational world, what is important to know and know how to do? The answer lies in identifying the knowledge and skills that reflect the demands and evolution of today's technologies, as well as capabilities that are uniquely human — and will remain so for the foreseeable future.

Computational thinking — a skillset for solving complex problems, a way to learn about topics across the curriculum, and a necessity for full participation in a computational world — is at the heart of what is important to know and know how to do.

In K-12 education, advocates for computer science are right to say that computational thinking is at the core of that discipline. It is also relevant across many other domains, joining cross-cutting concepts like global competence, social and emotional intelligence, and critical thinking. As K-12 schools expand computer science access, computational thinking should also be highlighted and integrated into multiple disciplines and provided for all students. Computational thinking is a critical gateway to full participation in a computational world and ensuring equity of access to opportunities to gain these skills is critical for a socially just and prosperous society.

## Acknowledgements

This report was developed under the guidance of Karen Cator, Colin Angevine, Josh Weisgrau, Chelsea Waite, and Jeremy Roschelle of Digital Promise. Susan A. Thomas supported the team with writing, editing, and advice.

The authors thank the experts interviewed for the content of this paper and those who reviewed and offered comments.

- Hina H. Baloch (General Motors)
- Karen Brennan (Harvard Graduate School of Education)
- Ryan Clarke (Girls Who Code)
- Leigh Ann DeLyser (CSforAll Consortium)
- Shuchi Grover (ACTNext)
- Mark Guzdial (Georgia Tech College of Computing)
- Sean Justice (Texas State University)
- Aileen Owens (South Fayette Township School District)
- Kylie Peppler (Indiana University Bloomington)
- Willa Peragine (Harvard Graduate School of Education)
- Gary Stager (Constructing Modern Knowledge)
- Chris Stephenson (Google)
- Pat Yongpradit (Code.org)

*This report was made possible by the generous support of General Motors.*

## Appendix A:
Leading K-12
Coding Non-Profits

# Black Girls CODE (BGC)

**Founded:** 2011

**Leadership:** Kimberly Bryant, Founder and CEO

**Headquarters:** Oakland, California and New York City

**Grade Levels:** Grades 6-12

## Mission/Vision

BGC's mission is to teach one million young and pre-teen girls of color how to code. BGC's vision is to increase the number of women of color in the digital space by empowering girls of color ages 7 to 17 to become innovators in STEM fields and builders of their own futures through exposure to computer science and technology.

## Main Programs

**CODE A Brighter Future.** Free hackathons for girls ages 12-17, designed to teach young black girls how to code and design mobile apps that help solve problems in their communities. Panels of judges — entrepreneurs, developers, and journalists — select winning teams whose members receive prizes and educational scholarships. Launched in 2017 in partnership with Colgate-Palmolive.

**Black Girls CODE Hackathons.** Hackathons for girls ages 12-17 that allow students to participate in creating solutions to social issues within their communities while they build their skills, confidence, and experience, and have fun.

## Measuring Impact

BGC has reached more than 8,000 young women in 14 chapters around the world. In 2017, Colgate-Palmolive became a sponsor of BGC's CODE a Brighter Future free hackathons. Also in 2017, General Motors teamed up with BGC to launch a Detroit chapter.

## Code.org

**Founded:** 2013

**Leadership:** Hadi Partovi, Co-founder and CEO; Alice Steinglass, President

**Headquarters:** Seattle

**Grade Levels:** K-12

### Mission/Vision

Code.org's mission is to expand access to computer science education in K-12 schools with a focus on increasing participation by women and underrepresented minorities. Code.org's vision is that computer science and programming will be a regular part of K-12 education, just like biology, chemistry, or algebra.

### Main Programs

**Hour of Code.** An annual campaign that has engaged 10 percent of all students in the world and provides the leading curriculum for K-12 computer science in the largest school districts in the United States. Hour of Code is a worldwide effort to celebrate computer science, starting with 1-hour coding activities but expanding to all sorts of community efforts.

Code.org also works with U.S. school districts to add computer programming courses to the core curriculum for K-12 students, and provides free online teaching and learning materials, including course and curriculum plans, online tutorials, and teacher trainings.

### Measuring Impact

In its online courses, 45 percent of students are girls, 48 percent are underrepresented minorities and 49 percent are on free or reduced meal plans. In high school classrooms, 37 percent are girls and 56 percent are African American or Hispanic. Hour of Code has served nearly 450 million (10 percent of students in the world), 49 percent of whom are female. More than 650,000 teachers have signed up to teach intro courses and more than 20 million students are enrolled. More than 120 of the largest school districts have partnered to add computer science to the curriculum—10 percent of all U.S. students and 15 percent of Hispanic and African American students. More than 20 U.S. states have changed policies to support computer science in K-12 education. Code.org courses are available in more than 50 languages and are used in 180+ countries.

## CodeNow

**Founded:** 2011

**Leadership:** Ryan Seashore, Founder and Chairman of the Board; Neal Sales-Griffin, CEO

**Headquarters:** New York City

**Grade Levels:** High school

### Mission/Vision

CodeNow's mission is to provide access to computer programming for students who do not have access to these learning opportunities. Its vision is to transform high school students into coders, designers, and product managers by teaching them to solve meaningful problems in their communities with software.

### Main Programs

**CodeNow 2.0.** A four-phase program model launched in 2017 that educates students through in-person and online learning experiences. The four phases are:

1. Weekend Workshops. 4 days (30 hours) of in-person instruction where participants learn how to go from an idea to a full-fledged, functional web application.

2. R&D Phase. Unlimited access to 60+ hours of self-paced, online learning and a workforce readiness program that includes in-person meetups and mentorship at partner tech companies. Students have opportunities to discover different roles within the tech industry.

3. HackNow Hackathons. 1–2 days (20 hours) of team building, in-person support, mentorship, and awards and prizes for competing amongst peers to build and launch a civic solution web application within 48 hours.

4. Summer Competition. 8–12 week (100 hours) national competition where teams of students create solutions utilizing product development, entrepreneurship, and advanced software engineering techniques to solve real problems affecting their lives and communities.

### Measuring Impact

CodeNow has taught more 2,000 underrepresented high school students how to code and engaged more than 500 volunteers for a total of nearly 13,000 volunteering hours.

## CSforAll Consortium

**Founded:** 2016

**Leadership:** Michael Preston, Co-Founder of CSforAll Consortium, and Executive Director of CSNYC

**Headquarters:** New York City

**Grade Levels:** High school

### Mission/Vision

The CSforAll Consortium's mission is to make computer science accessible to K-12 students and encourage computer science education in K-12 schools. The Consortium serves as the national hub of the Computer Science for All movement, which works to enable all students in grades K-12 to achieve computer science literacy as an integral part of their educational experience, both in and out of school.

### Main Programs

The CSforAll Consortium sets a collective agenda together with its membership of content providers, education associations, researchers, and supporters to help schools and districts provide all students with rigorous K-12 computer science education. The Consortium serves as a platform for connecting diverse stakeholders, providing support to new and developing initiatives, tracking and sharing progress, and communicating about the work to local and national audiences.

Membership comprises organizations around the country that share the mission of making computer science accessible to K-12 students, and includes organizations, researchers, and funding organizations. CSforAll hosts annual summits to bring together leaders in K-12 computer science education and like-minded organizations to share their progress and discuss new initiatives.

Initial funding was provided by the National Science Foundation (NSF).

### Measuring Impact

In 2016–2017, more than 5 percent of schools nationally participated in a program offered by the members of the CSforAll Consortium. The consortium has more than 400 members that have provided nearly 200 opportunities to learn and represent 39 U.S. states and territories.

## CSNYC

**Founded:** 2013

**Leadership:** Michael Preston, Executive Director

**Headquarters:** New York City

**Grade Levels:** K-12

### Mission/Vision

CSNYC's mission is to ensure that all of New York City's 1.1 million public school students have access to a high-quality computer science education that puts them on a pathway to college and career success.

### Main Programs

CSNYC's four areas of program development are as follows:

1. Community Building. CSNYC is committed to building communities through multiple sectors and stakeholders committed to K-12 computer science education.

2. Industry Engagement. CSNYC supports interaction with and exposure to the tech industry for students through job shadowing, internships, site visits, and more, to inspire future careers.

3. Teacher Pipeline. CSNYC sponsors programs to ensure the sustainability of the supply of teachers able to teach K-12 computer science.

4. Research. CSNYC supports K-12 students engaging in computer science research with professionals and the broader CS research community.

In addition, CSNYC oversees the national CSforAll Consortium (see separate profile in this appendix).

### Measuring Impact

In 2015, CSNYC partnered with the City of New York to launch CS4All, which has enabled 5,000 teachers to be able to teach computer science in almost 250 elementary, middle, and high schools in New York. In 2016, CSNYC launched the CSforAll Consortium, which will keep track of the impact of country-wide initiatives of coding in K-12 schools.

## Exploring Computer Science (ECS)

**Founded:** Project evolved out of the Computer Science Equity Alliance (CSEA), founded in 2004

**Team:** Gail Chapman, Julie Flapman, Joanna Goode, John Landa, Jane Margolis, Solomon Russell, Jean Ryoo, Todd Ullah

**Location:** Los Angeles

**Grade Levels:** High school

### Mission

ECS is a K-12/university partnership funded by the National Science Foundation (NSF) whose mission is to increase and enhance the computer science learning opportunities in the Los Angeles Unified School District (LAUSD), the second largest school district in the country, and broaden the participation of African-American, Latino/a, and female students in learning computer science.

### Programs

**Curriculum.** Uses an inquiry-based instruction model where students are guided through small group learning activities, exploration, role playing, and creativity. The curriculum is a yearlong course for high schoolers that covers the following major topics and areas: human computer interaction, problem solving, web design, programming, computing and data analysis, and robotics.

**Professional Development.** A professional learning community of teacher leaders that consists of a summer institute focused on both course content and pedagogical knowledge, along with ongoing PD and inquiry groups throughout the year, and an in-classroom coaching program.

### Measuring Impact

ECS has grown from LAUSD to a national level program as the curriculum and PD programs for teachers have been implemented in many U.S. states. ECS is supported and carried out in the seven largest school districts in the country. Overall, ECS has served over 8,000 high school students, and over 70 percent reported “liking” or “loving” the ECS curriculum. The demographic breakdown of student enrollment from the 2013–14 school year was almost 50 percent female, 70 percent Latino/a, 10 percent African American, 7 percent white, 6 percent Asian, and just under 4 percent Filipino and Native American.

## Girls Who Code

**Founded:** 2012

**Leadership:** Reshma Saujani, Founder and CEO

**Headquarters:** New York City

**Grade Levels:** Grades 6-12

### Mission/Vision

Girls Who Code's mission is to close the gender gap in tech, one girl at a time. Its vision is a future where the next generation of girls and boys prosper through creativity, bravery, and teamwork.

### Main Programs

Girls Who Code is building the largest pipeline of future female engineers in the United States by offering learning opportunities for students and alumnae to deepen their computer science skills, as well as their confidence; creating clear pathways for alumnae from middle and high school into the computing workforce; building a supportive sisterhood of peers and role models who help students and alumnae persist and succeed.

**Clubs Program.** After-school clubs for 6-12th grade girls to explore coding and programming, often led by volunteers. The clubs take place across the United States and in most major cities.

**Summer Immersion Program.** 7-week summer programs for 10-11th grade girls to gain serious programming skills and experience and get exposure to tech jobs.

### Measuring Impact

Girls Who Code has programs and clubs in every U.S. state and has partnered with major tech companies. By the end of 2017, 40,000 girls will have been involved in its programs, with 55 percent in high school, 37 percent in middle school, and 8 percent in college. Of these, 65 percent of students participating in its clubs and 93 percent of summer immersion program students say they are interested in majoring in computer science because of Girls Who Code.

## ScriptEd

**Founded:** 2012

**Leadership:** Maurya Couvares, Founder and Executive Director

**Headquarters:** New York City

**Grade Levels:** High school

### Mission/Vision

ScriptEd equips students in under-resourced schools with the fundamental coding skills and professional experiences that together unlock potential and create access to careers in technology.

### Main Programs

**Programming Courses.** Yearlong foundational and advanced courses in programming in partner schools, taught on a volunteer basis by software developers. For students who complete the advanced courses, ScriptEd matches students with paid summer internships in local tech companies.

**Hackathons.** Hackathons for students to work together on projects and showcase their skills.

### Measuring Impact

In the 2015–2016 school year, ScriptEd was in 31 high schools in New York City and served over 600 students. In 2017, ScriptEd began a pilot program in San Francisco. In the 2014–2015 school year, ScriptEd's student population was 43 percent African American, 30 percent Hispanic, 22 percent Asian, and 3 percent White. In addition, 36 percent of students were female; 87 percent of our students qualified for free or reduced-price lunch.

# Appendix B:
## Digital Promise Computational Thinking Educator Micro-Credentials

Digital Promise has built an innovative system of micro-credentials to recognize educators for the skills they learn throughout their careers in order to craft powerful learning experiences for their students. Each micro-credential in Digital Promise's system is:

- **Competency-based:** Micro-credentials allow educators to focus on a discrete skill related to their practice.
- **On-demand:** Through an agile online platform that clearly identifies each micro-credential's competency and required evidence, educators can start and continue the process of earning micro-credentials on their own time.
- **Personalized**: Because educators are able to select the micro-credentials they wish to earn, they can create their own professional learning journey aligned to their specific student needs and school-wide instructional goals.
- **Shareable**: Once educators earn micro-credentials, they can display the digital badges on Edmodo, LinkedIn, their CV/résumé, or a blog to signal their demonstrated competence wherever their professional journey might take them.

Grounded in research on computational thinking and learning sciences, Digital Promise has created 10 educator micro-credentials recognizing the key elements and pedagogical practices of computational thinking.

- **Computational thinking: key elements**
  - **Working with data**. Educator supports student inquiry practices using data to investigate questions and communicate findings.
  - **Creating algorithms**. Educator supports students in using algorithmic thinking to formulate procedures as algorithms and compare different solutions to the same problem.
  - **Understanding systems with computational models**. Educator supports students in developing systemic understandings of concepts by engaging with computational models.
  - **Creating computational models**. Educator supports students in using computational thinking to model the behavior of a system that has interrelated parts.
  - **Developing computational literacies**. Educator supports students in understanding and participating in computational literacies.

- **Computational thinking: pedagogical practices**
  - **Creating an inclusive environment for computational thinking.** Educator cultivates a learning environment that provides students opportunities to build knowledge and express themselves through computational thinking.
  - **Integrating computational thinking into curriculum.** Educator supports students in using computational thinking to develop understandings of ideas central to a discipline.
  - **Assessing computational** thinking. Educator uses assessment feedback to support student growth in computational thinking.
  - **Using computers as tools for thinking.** Educator documents and analyzes the ways students use computers as tools for representing their thought processes and connecting their learning to that of their peers.
  - **Selecting appropriate tools for computational thinking.** Educator selects computational tools that provide the appropriate support to meet computational thinking learning goals for diverse students.

View these micro-credentials and others in Digital Promise's ecosystem by visiting www.digitalpromise.org/bloomboard.

# References

1 Sanders, L. (2017). Smartphones may be changing the way we think. Retrieved from https://www.sciencenews.org/article/smartphones-may-be-changing-way-we-think

2 Tucker, A., Deek, F., Jones, J., McCowan, D., Stephenson, C., & Verno, A. (2003). A model curriculum for K–12 computer science. *Final Report of the ACM K-12 Task Force Curriculum Committee, CSTA*.

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

4 World Economic Forum. (2016). The Future of Jobs. Retrieved from http://reports.weforum.org/future-of-jobs-2016/

5 Rifkin, J. (2016, January 14). Davos Conference Misfires On 2016 Theme. Retrieved from https://www.huffingtonpost.com/jeremy-rifkin/the-2016-world-economic-f_b_8975326.html

6 Berger, T & Frey C. (2016). Structural Transformation in the OECD: Digitalisation, Deindustrialisation and the Future of Work. *OECD Social, Employment and Migration Working Papers No. 193*. Retrieved from http://www.oecd-ilibrary.org/social-issues-migration-health/structural-transformation-in-the-oecd_5jlr068802f7-en

7 Smith, J. (2017, August 3). A Robot Can Be a Warehouse Worker's Best Friend. *WSJ Online*. Retrieved from https://www.wsj.com/articles/a-robot-can-be-a-warehouse-workers-best-friend-1501752600

8 Chandarana, D., Faridi, F., Moon, J., & Schulz, C. (2017, July). How cognitive technologies are transforming capital markets. *McKinsey & Company.* Retrieved from https://www.mckinsey.com/industries/financial-services/our-insights/cognitive-technologies-in-capital-markets

9 Kanellos, M. (2016, January 13). Hold The Laughter: Why The Smart Fridge Is A Great Idea. Retrieved from https://www.forbes.com/sites/michaelkanellos/2016/01/13/hold-the-laughter-why-the-smart-fridge-is-a-great-idea/#648442437d40

10 IBM. (n.d.). Watson. Retrieved from https://www.ibm.com/watson/

11 DeepMind. (n.d.). AlphaGo. Retrieved from https://deepmind.com/research/alphago/

12 Stanford Robotics Lab. (n.d.). Ocean One Lands on the Moon. Retrieved from http://cs.stanford.edu/group/manips/ocean-one.html

13 Code.org. (n.d.). Leaders and trend-setters all agree on one thing. Retrieved from https://code.org/quotes

14 Della Cava, M. (2015, August 20). Should students learn coding? Students, schools disagree, poll finds. *USA Today*. Retrieved from https://www.usatoday.com/story/tech/2015/08/20/google-gallup-poll-finds-parents-want-computer-science-education-but-administrators-arent-sure/31991889/

15 Papert, S. & Solomon, C. (1971). Twenty Things to Do with a Computer. Retrieved from http://www.stager.org/articles/twentythings.pdf

16 Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. New York, NY, USA: Basic Books, Inc.

17 Scratch. (n.d.). Retrieved from https://scratch.mit.edu/

**18** MIT Media Lab. (n.d.). Lifelong Kindergarten: Engaging people in creative learning experiences. Retrieved from https://www.media.mit.edu/groups/lifelong-kindergarten/overview/

**19** Gilpin, L. (2014, December 8). How 'Hour of Code' Sparked a Movement That Could Teach 100 Million People to Code. *Tech Republic*. Retrieved from https://www.techrepublic.com/article/how-hour-of-code-sparked-a-movement-that-could-teach-100-million-people-to-code/

**20** Hour of Code. (n.d.). Retrieved from https://hourofcode.com/us

**21** Burning Glass Technologies. (2016, June). Beyond Point and Click: The Expanding Demand for Coding Skills. Retrieved from http://burning-glass.com/research/coding-skills/

**22** Dewey, J. (2007). *Experience and education*. Simon and Schuster.

**23** Kafai, Y. & Margolis, J. (2014, October 7). Why the 'coding for all' movement is more than a boutique reform. *Washington Post*. Retrieved from https://www.washingtonpost.com/news/answer-sheet/wp/2014/10/17/why-the-coding-for-all-movement-is-more-than-a-boutique-reform

**24** Code.org. (n.d.). About us. Retrieved from https://code.org/about

**25** Black Girls Code. (n.d.). Retrieved from http://www.blackgirlscode.com/

**26** Girls Who Code. (n.d.). Retrieved from https://girlswhocode.com/

**27** Code Now. (n.d.). Retrieved from https://www.codenow.org/

**28** Next Generation Science Standards. (n.d.). Retrieved from https://www.nextgenscience.org/

**29** World Economic Forum. (2016). The Future of Jobs. Retrieved from http://reports.weforum.org/future-of-jobs-2016/

**30** National Research Council. (2011). Report of a Workshop of Pedagogical Aspects of Computational Thinking. Retrieved from http://people.cs.vt.edu/~kafura/CS6604/Papers/NRC-Pegagogy-CT.pdf

**31** Common Core State Standards Initiative. (n.d.). Retrieved from http://www.corestandards.org/

**32** Tizon, C. (2016, March 9). Undergraduate Degree Earners Report, 2014-2015. *National Student Clearinghouse Research* Center. Retrieved from https://nscresearchcenter.org/undergraduatedegreeearners-2014-15/

**33** Computing Research Association. (2017). Generation CS: Computer Science Undergraduate Enrollments Surge Since 2006. Retrieved from https://cra.org/data/Generation-CS/

**34** Google Inc. & Gallup Inc. (2016). Diversity Gaps in Computer Science: Exploring the Underrepresentation of Girls, Blacks and Hispanics. Retrieved from http://goo.gl/PG34aH

**35** Georgia Tech School of Computer Science. (2016, December 7). Analysis of 2017 AP Computer Science Testing Reveals Ongoing Need for Qualified High School Teachers. Retrieved from http://www.scs.gatech.edu/news/584765/analysis-2016-ap-computer-science-testing-reveals-ongoing-need-qualified-high-school

**36** Code.org. (2017). 2007-2017 AP Computer Science Exam Results. Retrieved November 22, 2017 from https://docs.google.com/spreadsheets/d/17wvXkEq95bRfsY6Sx-Ilk8XxKPk8cutho6c7JyJP1UM/edit#gid=0

**37** The United Nations Office for Disaster Risk Reduction. (2015). Women's Leadership in Risk-Resilient Development. Retrieved from http://www.unisdr.org/files/42882_42882womensleadershipinriskresilien.pdf

**38** Phillips, K. (2014, October 1). How Diversity Makes Us Smarter. *Scientific American.* Retrieved from https://www.scientificamerican.com/article/how-diversity-makes-us-smarter/

**39** Code.org. (n.d.). Nine Policy Ideas to Make Computer Science Fundamental to K-12 Education. Retrieved November 22, 2017 from https://code.org/files/Making_CS_Fundamental.pdf

**40** Code.org. (n.d.). State Tracking 9 Policies. Retrieved November 22, 2017 from https://docs.google.com/spreadsheets/d/1YtTVcpQXoZz0IchihwGOihaCNeqCz2HyLwaXYpyb2SQ/pubhtml

**41** US Department of Education Office of Career, Technical, and Adult Education. (2016, December). Expanding Computer Science Education with Career and Technical Education. Retrieved from https://sites.ed.gov/octae/2016/12/19/expanding-computer-science-education-with-career-and-technical-education/

**42** Computer Science Teachers Association. (n.d.). About the CSTA K-12 Computer Science Standards. Retrieved from https://www.csteachers.org/page/standards

**43** K-12 Computer Science. (n.d.). K-12 Computer Science Framework. Retrieved from https://k12cs.org/

**44** CS for All Consortium. (n.d.). Retrieved from http://www.csforall.org/

**45** Smith, M. (2016, January 30). Computer Science for All. *The White House, President Barack Obama.* Retrieved from https://obamawhitehouse.archives.gov/blog/2016/01/30/computer-science-all

**46** CSNYC. (n.d.). Retrieved from https://csnyc.org/

**47** Gallup Inc. (2016). Pioneering Results in the Blueprint of U.S. K-12 Computer Science Education. Retrieved from http://csedu.gallup.com/home.aspx

**48** United States Department of Education. (n.d.). 2016 Title II Reports: National Teacher Preparation Data. Retrieved November 22, 2017 from https://title2.ed.gov/Public/Home.aspx

**49** K-12 Computer Science. (n.d.). Navigating the Practices. Retrieved from https://k12cs.org/navigating-the-practices/

**50** Christo. (2014). The Floating Piers. Retrieved from http://christojeanneclaude.net/projects/the-floating-piers

**51** Denning, P. (2017). Computational Thinking in Science. *American Scientist*. Retrieved from https://www.americanscientist.org/article/computational-thinking-in-science

**52** Bundy, A. (2007). Computational thinking is pervasive. *Journal of Scientific and Practical Computing*, *1*(2), 67-69.

**53** Cambridge Assessment. (2013, January). What is literacy? An investigation into definitions of English as a subject and the relationship between English, literacy and 'being literate'. Retrieved from http://www.cambridgeassessment.org.uk/Images/130433-what-is-literacy-an-investigation-into-definitions-of-english-as-a-subject-and-the-relationship-between-english-literacy-and-being-literate-.pdf

**54** Kafai, Y. B. (2016). From computational thinking to computational participation in K-12 education. *Communications of the ACM*, *59*(8), 26-27.

**55** diSessa, A. A. (2000). *Changing minds: Computers, learning, and literacy.* Cambridge: MIT Press.

**56** Vee, A. (2013). Understanding computer programming as a literacy. *Literacy in Composition Studies*, *1*(2), 42-64.

**57** Vee, A. (2017). *Coding Literacy: How Computer Programming is Changing Writing*. MIT Press.

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

**59** Reach Capital. (2017, August). Field Report on K12 Computer Science. Retrieved November 22, 2017 from https://drive.google.com/file/d/0B2eCjHNmaBGZeGpwTGlRTUJKZlU/view

**60** Private conversations with computer science education researchers and educators