Isidore Newman School

Computer Science at Newman

A comprehensive investigation and implementation strategy for K-12 coding

Jenna deBoisblanc

March 2016

Acknowledgments

Oskie Creech.

Xiao Xiao.

Chris Bigheno

FWCD

Susie

Foreword

While I strongly support the axiom that scientific research should eschew first person pronouns, I'd like to offer a foreword encapsulating my own experiences with computer science, a narrative that I believe is highly relevant in a document examining the importance of K-12 computer science education.

Coding is my greatest passion and my greatest asset. I've coded at every step of my career, from lab equipment for my physics thesis, databases for software companies, pollution data for environmental non-profits, and the list goes on. While my classmates struggled to find employment upon graduating in 2011 (the end of the recession), I immediately landed a cushy software job complete with a beer fridge and catered lunches. But perhaps most importantly: I code in my free time for *for pleasure* because coding is an incredibly versitile creative platform.

And yet, upon graduating from high school, I had no idea what computer science entailed or that software engineering might be a career I'd later consider persuing. If I had, I would have majored, or at least minored, in CS. In addition, I've written at length about the reasons why, *as a woman*, I almost didn't major in physics in college. I'll suffice it to say, closing the gender gap in hard sciences begins in elementary and secondary education.

Fortunately, I lucked out and graduated with a degree that mandated ample coding experience, and I consider myself highly priviledged to have found a subject that affords such opportunity. I hope to use my experiences not as a bias propelling a blind campaign in search of the latest trend, but rather as the bedrock passion behind the push to ensure that education keeps pace with evolving tools and means of exploration.

Onwards!

Jenna deBoisblanc

A Self-Consistency Note...

This document was coded with the programming language $\LaTeX$.

Contents

Chapter 1

Introduction

1.1 What is computer science?

For the purpose of avoiding potentially derailing semantics, it's instructive to begin with definitions of computer-related concepts used throughout this document.

"Coding" is the latest term to gain momentum in the education vernacular, popularized by the Hour of Code and online learning platforms like Scratch. While "coding" may be used to convey the beginning steps of **computer programming** [1], the terms are used synonymously and can be defined as follows: the process of writing or compiling machine instructions to accomplish a task. Coding is most commonly associated with the creation of websites, software applications, and mobile apps; however, there are innumerable potential applications of code. There are languages to program music, wearable tech, autonomous drones, interactive art visualizations, financial data, research documents (like this one!), 3D printers, robotics, and the list goes on.

Perhaps less common but equally as important is the concept of **"computational thinking"** (CT). Jeanette Wing, the Director of the Carnegie Mellon Center for Computational Thinking, is arguably the projenitor of the term [2]. The center describes CT as "a way of solving problems, designing systems, and understanding human behavior that draws on concepts fundamental to computer science... [by] making use of different levels of abstraction and thinking algorithmically" [3]. Google for Education defines CT as "a problem solving process that includes a number of characteristics, such as logically ordering and analyzing data and creating solutions using a series of ordered steps (or algorithms), and dispositions, such as the ability to confidently deal with complexity and open-ended problems" [4]. To summarize, CT is a set of problem-solving mindsets independent of computer programming that are applicable to all disciplines, not just computing. Emphasis on CT, especially in early K-12 education, will lay a strong foundation for future coursework in a myriad of fields.

Computer science (CS) employs computational thinking mindsets in the study and execution of computer programs and algorithmic processes. Coding is an essential, but not exclusive, component of CS, as is the study of hardware and software designs, their applications, and their impact on society.

While coding, computational thinking, and computer science are used interchangably throughout this document to refer to a set of K-12 curriculum encompassing all three, is important to distinguish "computer science" from **"computer literacy."** The National Center for Women and Information Technology states, "the ability to create and adapt new technologies distinguishes computer science from computer literacy, which focuses more on using existing technologies (e.g., word processing, spreadsheets)" [5]. Computer science teaches students how to think logically, solve hard problems, and understand the internal processes of computing rather than merely understanding how to use specific software. Although this document will focus primarily on CS, computer literacy is neither obsolete nor irrelevant (e.g. keyboarding) in 21st century education.

1.2 The importance of coding in K-12 education

In September 2015, New York City Mayor Bill de Blasio announced that all of the city's public schools would be required to offer computer science within the next 10 years. The announcement was neither the first of its kind nor the boldest; in December 2013, Chicago Mayor Rahm Emanuel launched Computer Science 4 All, a plan mandating a year-long computer science high school graduation requirement by 2019 [6]. And on the national stage, in early December 2015 President Obama signed into law a bipartisan bill - the Every Student Suceeds Act - recognizing computer science as a foundational academic subject on equal footing with subjects like math and English [7]. Municipal, state, and federal policymakers' recent focus on computer science education is no surprise. Education, especially K-12 education, is failing to keep pace with one of the most powerful tools of our time, jeopardizing the success and employment potential of American students.

A NEW LITERACY

In the rush by policymakers and educators to bring computer science to K-12 education, the term "coding literacy" has risen in the educational vernacular. While there are strong arguments for treating computer science as a "core" subject that permeates all levels of primary and secondary education, it's important to dissect the buzzword and extract the true foundational skills relevant to all students.

In a post by Chris Granger titled, "Coding is Not the New Literacy," Granger equates coding to pens and pencils - tools, not fundamental skills, employed in the expression of a literacy. Just as reading comprehension and writing composition - processes of solidifying cogent thoughts

- form the true basis of literacy, computational thinking mindsets and "modeling," Granger argues, form the basis of this "new literacy." He defines modeling as, "creating a representation of a system (or process) that can be explored or used."

In an age where programming languages and associated syntaxes, libraries, APIs, frameworks, hardware, and databases are rapidly-evolving, the true value of a computer science education lies not in an understanding coding mechanics or "computer literacy" (as defined in the previous section). It is the ability to think computationally and algorithmically, to develop and model systems, to decompose and debug complex problems, to explore and verify hypotheses rapidly and virtually: for these reasons, computer science is paramount in education.

Computer science happens to be an extremely powerful vector for modeling in almost any imaginable discipline. Just to list a few examples: modeling weather and climate patterns to predict the impacts of climate change, evaluating stress and strain of engineered structures, developing 3D animations for video games and entertainment, using image recognition algorithms to better detect breast cancer, creating financial models to measure the volatility in markets, or even, as researchers at the University of Maryland have shown, predicting the time and location of insurgent attacks in the Middle East[8].

Computer science understood as modeling gives teachers a way to make every subject hands-on and concrete. Grady Simon, a Project Manager at Microsoft writes,

> English classes teach models of English grammar, models of what it means to write well, models of narratives and arguments, and then they have students practice using those models to understand and write the English language. Computer science is the domain of pure models. The grammar of English sentences can be modeled as trees [a computer science data structure]. Arguments also tend to have a tree structure. Computer science talks about the properties of trees and grammars in general, among many other things.
>
> I think this is even more obvious in more technically rigorous subjects like science. Every scientific theory is a model, and because those models tend to be precise and rigorously defined, usually they lend themselves well to being instantiated as computer programs. Imagine a physics class where each homework assignment had students add a feature to a physics simulator, implementing in code the rules of physics they just learned about in class. Implement the laws of objects moving through space under gravitational field. Then you can have students play with the models. What happens if the gravitational constant were negative? What would it actually look like if gravity just imparted constant velocity downward instead of constant acceleration?

No matter what field of study students choose to persue, a solid foundation in computer science offers powerful ways to model, understand, and engage with hard problems.

CREATIVITY AND INNOVATION

To ignore the creative possibilities of coding is to ignore one of the most robust mediums of creative expression currently available. There are a myriad of programming tools for creating and communicating both physical and virtual ideas. Processing, a visual programming language designed for artists, is increasingly used to introduce programming thanks to its easy-to-learn platform and emphasis on the creation of art. “Generative” or autonomous virtual kaleidoscopes, interactive data visualizations, or colorful animated stories are just a few possibilities with an introductory-level knowledge of coding.

Max 7 is a block-based programming language for programming music, videos, and physical media. Innovative interfaces for synthesizing new sounds, video projections triggered by a dancer’s body movements, and interactive LED light shows controlled by live music are just a few of the infinite possibilities.

Video game design and 3D animation enables artists to code worlds not bound by the contraints of reality. As virtual reality gains momentum thanks to cheap interfaces like Google Cardboard, there will be even greater opportunities for the creation of 3D programmed worlds.

But the power of creative coding doesn’t fall strictly into the domain of art. Research has shown that some of **the most effective learning experiences come when we make or design things**, especially those that have relevance to our lives or the lives of people around us [9]. Computers can be viewed as a “universal construction material” granting rich opportunities for teaching or learning a diversity of subjects [10].

Mitchel Resnick, the director of the Lifelong Kindergarten group at the Massachusetts Institute of Technology’s Media Lab, is an expert on the use of coding as a creative medium for early education. He led the development of Scratch, a block-based coding platform that allows kids to create and share their own interactive stories, games, and animations. Resnick writes, “Coding is not a set of technical skills but a new type of literacy and personal expression, valuable for everyone, much like learning to write. We see coding as a new way for people to organize, express, and share their ideas” [11].

The creative potential of code has significant implications for entrepreneurial innovation as well as creative expression. In “Rethinking Learning in the Digital Age,” Resnick states:

> The proliferation of digital technologies has accentuated the need for creative thinking in all aspects of our lives, and has also provided tools that can help us improve and reinvent ourselves. Throughout the world, computing and communications technologies are sparking a new entrepreneurial spirit, the creation of innovative products and services, and increased productivity. The

> importance of a well-educated, creative citizenry is greater than ever before [12].

Coding is 21st century's universal tool for making, designing, and communicating. As technology rapidly evolves, so much our modes of innovating and creating.

COMPUTER SCIENCE EMPLOYMENT

Politicians on all levels of government are attempting to answer the technology industry's clarion call to fill a growing shortage of engineers.

According to the U.S. Bureau of Labor Statistics, computer and information technology jobs will be some of the fastest growing occupations in America, making up approximately 2/3 of the 1.1 million new STEM jobs projected for 2012-2022. [13]. By 2022 the Bureau of Labor Statistics estimates there will be over 1 million open computing jobs [13]. Code.org, a non-profit dedicated to expanding computer science education, emphasizes that these computing jobs will fall under every industry, not just tech. Finance, business, government, manufacturing, and education are all industries projected to need computating skills in the next decade [14].

Not only are unfilled computer science jobs abundant, they have high income potential. The highest paying degree in the U.S. is a computer science degree from Carnegie Mellon ($84,000), and among all universities, the National Association of Colleges and Employers (NACE) survey ranks computer science as first or second in income potential [15].

WORKFORCE SHORTAGE

Despite the high growth projections of well-paying computing jobs, skilled programmers aren't entering the workforce at a rate necessary to keep up with demand. Code.org estimates that there are 600,000 computing jobs open in the U.S. but only 38,175 students graduated with a computer science degree in 2013 [14]. 38,175 represents under 8% of all STEM degrees earned in 2013, a problematic number considering that projections estimate 2/3 of all STEM jobs will be in a computing field [13].

K-12 EDUCATION GAP

Education, especially K-12, has a significant role to play in the growing shortage of computer sciene graduates. Only 1 in 4 U.S. schools offer computer science classes with computer programming [16], and in 23 states computer science cannot be used to meet math or science graduation requirements [7]. According to the Gallup study [16], many school administrators do not perceive a demand by parents and students for computer science. The study states that the main reason school principals do not offer computer science is the lack of time for classes not required by testing, as well as the inability to hire computer science teachers due to availability and budget constraints [16].

Group	**AP CS Exam**	**CS Bachelor's**	**Computing Jobs**
Females	22%	17%	23%
People of Color	13%	18%	14%

TABLE 1.1: Percentage of Underrepresented Groups in Computer Science [14]

UNDERREPRESENTED GROUPS

Women and people of color are especially at risk of falling behind and losing access to high-paying computing employment. In 2011, women made up 21% of students who took the AP computer science exam, and less than 1% were African American [2]. For many students, access to technology is the greatest hurdle. Hispanic students have less access to internet access at home than white or black students [16].

In the context of evaluating the importance of K-12 computer science education, it is critically important to point out that this opportunity gap begins in K-12 education. Table 1.1 indicates that by high school, women and people of color are underrepresented in the AP CS exam; this trend remains relatively stable through college and into the workforce.

While closing the gender and achievement gap may seem daunting, early and frequent exposure to computer science throughout K-12 education has been demonstrated to have significant impacts on career outcomes. Women who try AP Computer Science in high school are ten times more likely to major in it in college, and Black and Hispanic students are seven times more likely [17].

CONCLUSION

Computing jobs are on the rise, and unless public policy and education quickly fall in line with the needs of an evolving workforce, American students will miss out on valuable opportunities to pursue high-paying careers in a myriad of fields. K-12 exposure to computer science, especially for women and people of color, is necessary to encourage students to explore computer science in college or beyond. But even for students whose futures eschew computing and technology, the benefits of computer science exposure and computational mindsets transcends scientific disciplines.

No matter what field of study students choose to persue, a solid foundation in computer science is one of the most powerful, flexible, creative, and innovative toolboxes for understanding and exploring our world in the 21st century.

Chapter 2

Research

2.1 Case Studies

2.1.1 Independent Schools

INDEX Schools

INDEX Schools is a collaboration of over 100 independent schools to "to share data, analysis, research, and information to aid member schools in decision-making, policy development, and strategic planning" [18]. At the beginning of the 2015 school year, the Assistant Head of Newman emailed the INDEX listserv to request any curriculum outlines or standards and benchmarks for PK-12 computer science that schools might be willing to share. The results of the query are captured in a Google Spreadsheet that can be accessed here:

Fort Worth Country Day School

Fort Worth Country Day (FWCD) is an independent, coeducational K-12 school in Fort Worth, Texas with a total enrollment of over 1110. Three Newman faculty members toured the school in February of 2016 to observe their CS program. Their Upper School CS sequence progresses as follows:

- 9th: Media Projects or AppInventor
- 10th: Art and Code with Processing
- 11th: AP CS A with Java
- 12th: Data Structures with Java

The US uses a block schedule. Classes meet for 75 minutes every other day for one semester. CS courses are treated as general electives, with the exception of "Art and Code," which satisfies a fine arts credit.

There is no US CS requirement at FWCD. Media Projects is currently one section, there are two sections of Art and Code, one section of AP CS, and beginning next year, one section of data structures. With the exception of the introductory course, all CS classes are covered by one faculty member with a computer science background.

"Art and Code" is a prime example of a course that emphasizes the creative capacity of coding. The course is taught in Processing, a visual programming language based on Java that is designed for artists. The teacher, Aaron Cadle, blends his own curriculum with content, labs, and projects developed by Darby Thompson's, a CS teacher at Sidwell Friends School in Washington D.C. who has developed a project-based introductory CS course. During the observation, Aaron tested student's understanding of abstract classes using a "Fish Tank" project. Students coded their own interactive, colorful fish that were placed on display in the library, and students outside of the course could vote on their favorite fish.

For AP CS Aaron relies on the A+ Computer Science curriculum, which includes worksheets, labs, tests, slides, and notes. During the observation, students pulled personally-relevant datasets from data.gov (baseball stats, Powerball numbers, Hilary Clinton emails, etc.) and explored meaningful ways analyze the data using Java (e.g. finding the most common Powerball numbers). Aaron reversed the typical AP structure (lecture in class, worksheets for homework) in order to devote more time to workshopping labs. Students watch short (5 min) instructional videos for homework and work on labs in class.

The upper-level CS course, Data Structures, is project based. The course is taught in Java and uses the XBox Kinect. Aaron uses the book, "Making Things See" by Greg Borenstein, which explores the use of Arduinos, Processing, and the Kinect to develop interactive programs.

As for Middle School, coding exposure is currently limited to a two week HTML project that is integrated into 7th grade science. A newly hired Middle School iPad Coordinator, brought on to organize the division's one-to-one iPad program, is working to bolster computer literacy across campus. She currently uses 5th and 6th grade advisory time to teach digital citizenship, email etiquette, and Google Classroom. She is also developing curriculum for a required MS technology/computer literacy elective that may be offered next year. When asked about curriculum standards, she pointed to ISTE benchmarks.

Lower School computer science exposure at FWCD is very similar to Newman's. A "Computer Special" offered to grades K-4 meets for 40 minutes every 1 out of 6 days. The class takes place in the "lab" - a proto-makerspace/ design thinking workshopping space. The special explores engineering exercises, programmable hardware (Sphero, LittleBits, Lego robotics, etc.), and coding (ScratchJr and Kodable). FWCD has never taught keyboarding; instead they direct interested parents/students to online resources. At one point the computer teacher, Mandy Lofquist, maintained a "technology checklist" of all the computing skills her students should acquire, although she hasn't relied on this list since her students started coding.

Greenhill

Greenhill School, a PK-12 coeducational private day school outside of Dallas, Texas, has an enrollment over 1200. This is another school that Newman faculty members toured in February, 2016 to observe their curriculum and classrooms. The Director of Technology, Chris Bigenho, has been researching computer science programs and developing K-12 CS curriculum for the last five years. Emphasis on aligning the program across divisions and with national standards has led to the current implementation, which is one of the more advanced programs in the INDEX schools consortium.

The Upper School CS program is a three-tier, two year sequence. Students may take multiple courses from each tier (each a semester long course). There is a 1 trimester CS graduation requirement. The courses include:

- CS1: Introduction to CS with Arduino
- CS1: Beginning JAVA Programming
- CS1: Introduction to Game Design
- CS1: Engineering
- CS2: Advanced Computational Design
- CS2: Intermediate JAVA Programming
- CS3: Advanced Topics in Computer (prepares students for AP)

Greenhill's Middle School courses are broken into trimesters and rotate every 6 days. Classes meet for approximately 50 minutes. One block of the schedule is split between two electives; each meets every other day corresponding to ACE or BDF blocks. Band and choir may meet during ACE the entire year, while 2D or 3D art meets for a single semester during BDF.

Greenhill offers two required electives, "Exploratory Design," that blend coding (Scratch, Arduino) with engineering design challenges. An example project includes prototyping, 3D printing, and testing a hub and blades for a wind turbine. The coding sequence is as follows:

- 5th grade - Exploratory Design (required)
- 6th grade - Exploratory Design (required)
- 7th/8th grade - Prototyping 1 (optional elective)
- 7th/8th grade - Prototyping 2 (proposed optional elective)
- 7th/8th grade - Game Design (proposed optional elective)

After Exploratory Design, the electives are no longer required. Prototyping I is an optional 7th and 8th grade "arts" elective that meets every other day for one trimester. Students work through "Arduino Projects," a project book of 10+ exercises that introduce Arduino, servos, LEDs, and other simple circuits.

Now that Greenhill has an introductory CS course at the US level, the plan (discussed by both the MS head and the director of technology) is to consider making a CS course required in 7th and 8th grades.

The LS exposure takes place in 2nd - 4th grades in Computer 2, 3, and 4. These courses are a mix of computer literacy and coding (Microworlds), which is likely to change now that the computer teacher is retiring.

2.1.2 Public High Schools

Examining the top fifteen public high STEM schools, as ranked by the U.S. News in 2015 [19], provides insight into the state of computer science in some of the nation's top STEM high schools. Roughly 3/4th of the list's top 15 schools offered at least one CS course, significantly more than the national average cited by Gallup of 1 in 4 U.S. schools [16]. About half of these schools offered a basic (no requirement, strictly elective) computer science sequence: an introductory CS course (frequently in Java), followed by the AP CS course. Four of the 15 schools had a robust, scaffolded CS sequence; these schools are examined below.

Thomas Jefferson High School for Science and Technology
Thomas Jefferson High School for Science and Technology (TJ) is ranked #2 on U.S. News' STEM report. A "regional Governor's school" in Virginia with 1,846 enrolled students, TJ offers five years of Computer Science. There is a one credit CS graduation requirement that "most students satisfy by taking Foundations of Computer Science in 9th grade during the summer" [20]. Additional courses (the sequence for which can be found on their website) include: AP Computer Science and Data Structures, Artificial Intelligence 1 & 2, Parallel Computing 1 & 2, Mobile App Development, Web App Development, and Mobile/Web Application Development Lab.

Thomas Jefferson HS's overall graduation requirements are listed in Table 2.1. Computer science courses, including the graduation requirement, fall under "electives." It's important to note that TJ's graduation requirements are largely similar to Newman's.

Stuyvesant High School
Stuyvesant High School, a nationally-ranked New York City public high school with 3,285 students, ranked 14th on the U.S. News' top STEM schools. Stuyvesant offers 6 active CS courses that are listed through the math department. Although none of the CS courses satisfy

Table 2.1: Thomas Jefferson High School Graduation Requirements [20]

Subject	**Required Credits**
English	4
Math	4
Science	4
History/Social Studies	4
World Language	3
Health/PE	2
Fine Arts	1
Economics/Personal Finance	1
Electives	3
Total	**26**

the 4-sequence math requirement, students are required to take a single semester "Introduction to Computer Science." In the introductory course,

> "Students will study some the basic themes and subfields of computer science including algorithms and programming, simulation, networking, computability, graphics, and artificial intelligence. Students will also be given a solid foundation in working in a modern networked computer environment. This is a required course beginning with the Class of 2007. Students will take the course either in the Fall or Spring term of their sophomore year" [21].

Additional CS courses offered to high school students include: AP Computer Science, System Level Programming, Computer Graphics, Software Development, and Computer Science Intel Research.

High Technology High School

High Technology High School (HTHS) in Lincroft, New Jersey, has an enrollment of 280 and is ranked #1 in U.S. News' top STEM schools. HTHS is relatively unique for its strong integration of technology and programming into engineering courses. CS courses include: Computer Programming for Engineers, Computer Science & Software Engineering, and Digital Electronics [22].

Edison Academy for Science, Mathematics, and Engineering Technologies

The Academy for Science, Mathematics, and Engineering Technologies, a magnet school located in Edison, New Jersey, has a total enrollment of 161 students and is ranked #4 in the U.S. News top STEM public high schools [19].

The high school operates on a rotating block schedule; all classes meet Monday for one period, and classes meet on either Tuesdy/Thursday or Wednesday/Friday for two periods (88 minutes). Engineering courses, however, meet every day of the week. Students enroll in either the "civil and mechanical" or "electronics and computer engineering" track. The requirements for the Electronics and Computer Engineering Technology (ECET) Program are listed in Table 2.2.

Table 2.2: Edison Academy: Electronics and Computer Engineering Technology (ECET) Program of Study [23]

9	Introduction to Engineering
	Introduction to Digital Logic
	Introduction to Computer Science Using C++
	DC Circuit Analysis
10	Integrated Circuit Logic Families
	Sequential Logic Circuits
	Finite State Machines (FSM)
	Interfacing to the analog world
	Microcontrollers / Assembly language
11	Object Oriented Programming Using C++
	AC Circuit Analysis
	Electronic Communication Systems
	Digital Communication Systems
12	Senior Project
	Programming with JAVA

Table 2.3: Four Year Plan for Westside HS Computing Sciences and Engineering [24]

9th	**10th**	**11th**	**12th**
English	English	AP Language	AP Literature
Algebra/Geometry	Geometry/ Algebra II	Alga. II/Pre-Calculus	Pre-Calculus/Calculus
World Geography/	World History	U S History	Go/Economics
Biology	Chemistry	Physics	AP Science/Elective Science
Foreign Language	Foreign Language	Foreign Language	Health/Speech
Magnet Course	Magnet Course	Magnet Course	Magnet Course
Fine Arts Credit	PE	Elective	Elective

Westside Magnet High School for Integrated Technology

Although Westside Magnet High School for Integrated Technology, did not make U.S. News' list of top 15 public STEM schools, its robust CS program warrants attention. The magnet school in Houston, Texas, gives students the option to choose one of five strands in business, computing sciences and engineering, fine arts, health science and technology, or media. The school reportedly uses the four year engineering curriculum developed by Project Lead the Way [24].

Students who choose the Computing Sciences and Engineering strand choose between one of three programs: Computer Programming, Computer Maintenance, and Engineering. Each program is an individually designed; a typical four year schedule is detailed in Table 2.3. "Magnet courses" (specific to the Computing Sciences and Engineering strand) are outlined in Table 2.4.

Table 2.4: Westside HS Computing Sciences and Engineering Magnet Courses [24]

9th	**10th**	**11th**	**12th**
Principles of Information Technology/ Principles of Art, A/V Technology and Communication	Pre-AP Computer Science 1 (Python) or Computer Progamming	Pre-AP Computer Science 2 (JAVA) or Pre-AP Computer Science 1 (Python)	AP Computer Science 1 (JAVA) or Pre-AP Computer Science 2 (JAVA)
Principles of Information Technology/ Principles of Art, A/V Technology and Communication	Computer Maintenance	Telecommunications and Networking	Research and IT Solutions or Practicum in Business Management
Introduction to engineering Design	Principles of Engineering	Digital Electronics	Engineering Drafting and Design
Concepts of Engineering	Robotics	Principles of Technology	Engineering Mathematics

2.1.3 Higher Education

Preparation for college is obviously paramount in any Upper School academic program at Newman, and so it's instructive to examine the range of CS courses and subjects offered in higher education. **MIT**
Massachusetts Institue for Technology is a consistently top-ranked, globally-recognized leader in education, research, and innovation. MIT's Electrical Engineering and Computer Science (EECS) Department is the largest department at MIT and is composed of four undergraduate degree programs: Electrical Science and Engineering, Electrical Eng. & Computer Science, Computer Science and Engineering, and Computer Science and Molecular Biology. All four tracks are required to take EECS I, an introductory electrical engineering and computer science course:

> **6.01 Introduction to EECS I** - An integrated introduction to electrical engineering and computer science, taught using substantial laboratory experiments with mobile robots. Key issues in the design of engineered artifacts operating in the natural world: measuring and modeling system behaviors; assessing errors in sensors and effectors; specifying tasks; designing solutions based on analytical and computational models; planning, executing, and evaluating experimental tests of performance; refining models and designs. Issues addressed in the context of computer programs, control systems, probabilistic inference problems, circuits and transducers, which all play important roles in achieving robust operation of a large variety of engineered systems [25].

There is a close link between electrical engineering and computer science throughout MIT's undergraduate CS program. Thanks to tools like Arduinos, Raspberry Pis, VEX and LEGO robotics, and other easily-programmed microcontrollers, it's possible to develop Upper School curriculum that explores computer science through an electrical or mechanical engineering lens. As for CS fundamentals, MIT's more traditional introductory CS course is described below:

> **6.S04 Special Subject: Fundamentals of Programming** - Introduces fundamental concepts of programming. Designed to develop skills in applying basic methods from programming languages to abstract problems. Topics include programming and Python basics, computational concepts, software engineering, algorithmic techniques, data types, and recursion and tail recursion. Lab component will consist of software design, construction and implementation of design [25].

Pomona College

Pomona College, a small liberal arts college in Claremont, California, offers the perspective of a top-tier post-secondary school without an engineering department. CSCI051 is the first course in the CS sequence, and it specifies that no previous programming experience is required.

> **CSI051** - Introduction to the field of computer science using the object-oriented language Java. Topics include iteration and recursion, basic data structures, sorting and searching, elementary analysis of algorithms and a thorough introduction to object-oriented programming. Special emphasis on graphics, animation, event-driven programming and the use of concurrency to make more interesting programs [26].

CSCI052, the second course in the introductory series, emphasizes "functional programming, procedural and data abstraction, recursion and problem-solving" as well as "computer architecture and organization, finite automata and computability" [26]. These subjects are closely aligned with the CSTA benchmarks for upper level computer science.

The following goals of the Pomona CS major have been pulled from the department's website: [27]

1. To conceptualize multiple views of problems, to develop computational solutions grounded in theory, and to evaluate their solutions using a range of metrics.
2. To work alone and in teams to identify, formulate, and solve computing problems.
3. To gain a firm grounding in the core areas of computer science: theory, systems, programming languages, and algorithms.
4. To apply the knowledge gained in core courses to a broad range of advanced topics in computer science, and to develop the ability to learn sophisticated technical material independently.

5. To be able to communicate technical information both orally and in writing.
6. To understand the theoretical, practical, and ethical ramifications of computational solutions to problems, and to be aware of current research developments in computer science.

2.2 National Standards

2.2.1 CSTA

The Computer Science Teachers Association (CSTA) released the revised "K-12 Computer Science Standards" in 2011, which is widely cited in the computer science education community as source of CS benchmarks. The CSTA outlines a set of learning objectives on three levels and grouped into one of five "strands": computational thinking; collaboration; computing practice; computers and communication devices; and community, global, and ethical impacts [28]. The benchmarks begin at the elementary level and continue through high school. AP CS curriculum fits well into the level 3 CSTA sequence, but CSTA recommends additional CS courses for depth. For the list of standards, refer to Appendix C.

2.2.2 ISTE

The International Society for Technology in Education (ISTE) has compiled a set of standards for computer science educators [29]. These standards are listed in Appendix D and are cross-referenced with CSTA benchmarks. Unlike CSTA, ISTE does not describe K-12 standards, but rather outlines a set of learning objectives that could presumably be covered in one or more advanced high school courses. When mapped to the CSTA benchmarks, ISTE standards mostly fall under level 3A (at the AP CS level) are are not particularly useful for identifying the requirements for introductory curriculum. In addition to computer science concepts, these standards layout teaching standards and strategies specific to educators and learning environments.

2.2.3 College Board's Advanced Placement

Since 2003 the Advanced Placement Computer Science A exam has tested students' ability to solve problems and design algorithms using Java. The College Board's test is intended to cover the equivalent of a first semester college computer science course. The topics include:

1. Object-Oriented Program Design: Program and class design
2. Program Implementation: Implementation techniques. Programming constructs, Java library classes

3. Program Analysis: Testing, Debugging, Runtime exceptions, Program correctness, Algorithm analysis
4. Standard Data Structures: Primitive data types, Strings, Classes, Lists, Arrays (1-D and 2-D)
5. Standard Operations and Algorithms: Searching, Sorting
6. Computing in Context: System reliability, Privacy, Legal issues and intellectual property, Social and ethical ramifications of computer use

In 2016 the College Board plans to release a second AP CS course, AP Computer Science Principles, that will operate in tandem to the other AP CS class. From the AP website, "AP Computer Science Principles offers a multidisciplinary approach to teaching the underlying principles of computation." Unlike the AP CS A which relies exclusively on Java, the Principles course allows teachers to use any programming language. The course introduces key programming concepts with a "focus on fostering students to be creative." The "Big Ideas" covered by this exam include:

1. Creativity
2. Abstraction
3. Data and Information
4. Algorithms
5. Programming
6. Internet
7. Global Impact

For a detailed list of learning benchmarks associated with this exam, check out the AP Computer Science Principles Curriculum Framework.

PROS & CONS OF AP CS

There are merits and shortcomings of teaching Advanced Placement courses in general, as well as those that are specific to computer science. Given the Perhaps the most salient advantage is the potential for college credit, which may allow students to place out of introductory-level college courses and save time or money in higher education. AP courses may also positively affect college transcripts since AP courses at Newman are weighted in the student's cumulative GPA calculation. In addition, because College Board is an internationally-recognized organization, there is a thriving community willing to offer support or share resources, curriculum,

AP Computer Science *A*	**AP Computer Science *Principles***
Curriculum:	
Focused on object-oriented programming and problem solving	Built around fundamentals of computing including problem solving, working with data, understanding the Internet, cyber-security, and programming.
Language:	
Java is the designated programming language	Teachers choose the programming language(s)
Skillset:	
Encourages skill development among students considering a career in computer science or other STEM fields	Encourages a broader participation in the study of computer science and other STEM fields, including AP Computer Science A
Assessment:	
Multiple-choice and free-response questions (written exam)	Two performance tasks students complete during the course to demonstrate the skills they have developed (administered by the teacher; students submit digital artifacts), and multiple-choice questions (written exam)

TABLE 2.5: Comparison of AP Computer Science Exams [30]

example labs, and exercises. Finally, the AP course offers a set of well-researched standards and benchmarks that, at in theory, align with equivalent college-level courses.

Perhaps one of the greatest criticisms of AP courses in general is the rigidity and inflexibility of the curriculum. John Tierney, a contributing writer for The Atlantic and a former professor of American government at Boston College, writes:

> "In short, AP courses are a forced march through a preordained subject, leaving no time for a high-school teacher to take her or his students down some path of mutual interest. The AP classroom is where intellectual curiosity goes to die" [31].

This shortcoming is particularly stifiling and problematic in AP CS A. Given the substantial amount of theory and subject matter covered on the exam, the course affords little opportunity for project-based, open-ended discovery and expression. The AP CS A course requires Java, a language that comes with significant paradigm baggage, and limits students to the kinds of programming that can be expressed with Java. Although Java is a common language in introductory college courses, the most common teaching language at the university level as of

July, 2014 is Python [32], a language that allows students "to focus less on details such as types, compilers, and writing boilerplate code and more on the algorithms and data structures relevant to performing their intended tasks" [33]. And although the curriculum provides a sound introduction to the theory and mindsets of CS, the AP CS A exam renders an infinitely creative subject into a formulaic and uninventive practice, which is not the most effective approach for engaging and capturing a diverse group of students.

The AP CS Principles course has the potential to mitigate many of these criticisms. The course website states that the impetus for developing the cours was the need to "increase the number of students interested in and prepared for success in computer science and other STEM fields" [34]. The course does not specify a specific programming language, but instead encourages students to develop "personally relevant artifacts" [35], whether these are web apps, digital music, animations, or mobile games. "The course is unique in its focus on fostering students to be **creative**" [35] (the use of bold font to emphasize "creative" comes straight from the document). There is also greater emphasis on appealing to a broader audience by highlighting the impacts of computational thinking on a myriad of fields, not just computer science.

On paper the new AP CS Principles course is the answer to the need for a college-level, nationally-recognized CS course that doesn't stifle innovation, creativity, and personalized, project-based learning. The course, however, is still young, and it may take time to answer questions such as, how will colleges respond? Is this course sufficiently rigorous preparation for college-level CS? Do the curriculum and benchmarks still afford space for creativity and exploration? Will students, particularly those from underrepresented groups, feel more inclined to take this new AP course?

As for enrollment, it's unclear how the AP designation might affect course registration at Newman. The AP CS Principles states that "students do not need to have prior knowledge of any programming language" [34]. Offering an AP as the introductory course may deter students who are unfamiliar with CS, particularly its creative dimension, from signing up for a rigorous course. Even if AP CS Principles is the second or third course in the CS series, there may be unpredictable impacts on registration. For example, high-achieving students who might typically take AP Biology or Chemistry electives may be more likely to take AP CS since it would offer the same GPA and transcript distinctions as other AP science or math courses, while students with a passion for computer science but who desire a less rigorous course load might shy away from an AP CS course. Another alternative- offering an AP caliber course that prepares students for an optional AP exam at the end of the semester- may cater to both groups of students. Ultimately, student feedback is necessary to determine the best course of action to maximize enrollment.

2.2.4 Next Generation Science Standards

The Next Generation Science Standards (NGSS) is the product of twenty-six states and broad-based teams working to develop a new set of science standards needed to revise a 15-year-old set of documents on which most state standards are based [36]. Unfortunately, the latest NGSS document does not enumerate standards specific to computer science, for the reasons explained below:

> "Computer Science and Statistics Computer science and statistics are other areas of science that are not addressed here, even though they have a valid presence in K-12 education... Computer science, too, can be seen as a branch of the mathematical sciences, as well as having some elements of engineering. But, again, because this area of the curriculum has a history and a teaching corps that are generally distinct from those of the sciences, the committee has not taken this domain as part of our charge. Once again, this omission should not be interpreted to mean that computer science or statistics should be excluded from the K-12 curriculum. There are aspects of computational and statistical thinking that must be understood and applied in learning about the sciences, and we identify these aspects, along with mathematical thinking, in our discussion of science practices in Chapter 3" [36].

Although the NGSS does not explicitly outline CS benchmarks, the Engineering Design standards are highly applicable to software engineering or computational thinking projects. These benchmarks are covered in Appendix E.

For CS courses that are not standalone but rather are integrated into existing math or science classes, aligning curriculum with relevant NGSS standards may be advisable. Code.org has published two sets of middle school curriculum- CS in Algebra and CS in Science- that are consistent with both CSTA and NGSS standards. A table summarizing the overlap between these standards as it relates to this curriculum can be found on their website.

2.3 Online Curriculum

A non-exhaustive list of online coding tools and curriculum can be found in Appendix A. Tools are separated by level (lower, middle, and upper school). The table contains information about professional development, platform cost, student tracking, and the possibility for self-guided online learning.

At the elementary school level, iPad apps (Kodable) and drag-and-drop visual programming languages (Hopscotch, Scratch, SNAP!, Tynker) teach foundational coding concepts through interactive games, art, and animations. Many of these platforms cater to teachers without computer science experience. For example, Tynker includes assessment, classroom management, lesson plans, and a built in tutor. In addition, Tynker aligns to Common Core Mathematics, Common Core ELA, and CSTA Computer Science standards [37].

Many of the elementary-level tools (Scratch, Tynker, SNAP!) can be used in more sophistocated projects at the middle school level. For integrating CS into existing middle school courses, Code.org provides 20-lesson CS in Algebra and CS in Science modules, as well as in-person or online professional development.

The upper school curriculum typically focuses on exploring computer science concepts with a true development language (Java, Javascript, Python, etc.). CodeAcademy and Kahn Academy use videos and cloud-coding to deliver instruction in a variety of languages, including Javascript, Ruby, Python, and more. Online AP Computer Science courses are available through CodeHS and Edhesive, each of which offer PD and student tracking.

ONLINE VS. CLASSROOM INSTRUCTION

Online instruction offers the ability to Online CS instruction, from Online Global Academy, an online platform offering courses in a variety of subject areas, currently offers self-motivated Newman students the ability to take the following CS courses:

- Art of Code: Processing
- Computer Programming I: Computational Thinking
- Computer Programming I: Java
- Computer Programming II: Advanced Java
- Computer Programming II: Analyzing Data with Python
- Computer Programming II: iOS Development

With a course you design yourself, you can recruit the best students regardless of background, and mold the curriculum to fit their needs and interests.

2.4 Hardware Tools

Physical coding platforms open the door to a variety of highly-engaging ways to teach coding to all ages. Robotics, wearable tech, electronic music, and interactive games are just a few examples of potential projects that can be used to teach coding or computational thinking. Please refer to Appendix B for a table of tools.

Chapter 3

Newman Past and Present

3.1 Previous Programs

The inclusion, dissolution, and subsequent reemergence of computer science in secondary education is not unique to Newman. As Kafai and Burke [2] point out, in the 1980s many schools featured Basic or Logo programming, but by the mid 90s, CS was on the decline. Kafai and Burke point to the lack of subject-matter integration, difficulty finding qualified instructors, and rise of the internet precipitating a shift of emphasis from programming to software-specific proficiency and web surfing literacy. They argue that several significant changes, such as the plethora of fun and engaging coding sites with self-guided, online curriculum, are responsible for a resurgance in CS education. In addition, they point to “a shift from tools to communities.” Sites like Scratch operates much like a social networking site that facilitates content sharing and open source exchange [2].

3.2 Present Status

3.2.1 Lower School

Coding Already Happening

Exposure to coding in Lower School currently takes place during Media. Susie Toso meets with grades 1-4 once per week for 50 minutes.

Definition 3.1.

2nd Code.org

3rd Scratch

4th Computer literacy: keyboarding (Typing Pal, which is currently the perview of the homeroom; 2x week for 10 minutes)

Jennifer Williams and Elaine Sevin offer a collaborative 2nd/5th grade LEGO WeDo robotics unit that meets once per month for 50 minutes. The excerise, to control a soccer player and kick a ball into a LEGO goal, simulates real life phenonmenon through LEGOs and code. Students are introduced to CS topics such as abstraction, problem deconstruction, and serialization of tasks. While this robotics exposure is a valuable and highly-engaging engineering exercise that clearly augments LS coding curriculum, additional time is required to convey foundational CS topics.

Finally, programmable hardware is used in 2nd grade science. During the Mars unit, students program Bee-Bots, small robots for teaching sequencing and problem solving.

LS Faculty Brainstorm

A facilitated brainstorm during an afternoon faculty meeting helped to elicit thoughts and concerns from all Lower School teachers regarding both computer literacy and coding. Faculty divided into groups based on a grade and answered the following questions on Post-its:

Definition 3.2.

Q1. How is technology used in your classroom?

Q2. What do you wish/ think your students should know that they currently don't about coding and computer literacy?

Q3. What are your hopes and dreams for computing?

Q4. What are your concerns about implementing coding curriculum at Newman?

A table of all of the responses can be found in Appendix [insert]. There were a handful of recurring ideas that emerged across all grades. LS teachers were concerned about the balance of coding vs. computer literacy, specifically that the rise of coding came at the cost of students' ability to effectively operate computers in class. Additional proficiency and knowledge of word processing, internet research, internet safety and ethics, file structure, keyboarding, and troubleshooting were some of the most common responses to questions two and three.

As for question four - concerns related specifically to implenting coding curriculum - teachers overwhelmingly cited the need for CS professional development. The issue of time was also raised multiple times: how will CS fit into the existing schedule, and what will be sacraficed to make room for this new subject?

Meeting Benchmarks

The CSTA benchmarks, found in Appendix C, are the most highly cited and comprehensive CS standards, especially for grades K-6. Many of the CSTA benchmarks are already met by Newman's existing curriculum. Discussed below are coding and computer literacy benchmarks less likely to be covered by default.

In grades K-2, CSTA places emphasis on computer literacy, multimedia exposure, and computational thinking. The ability to break large problems into discrete steps, to think sequentially, and to sort information are the core of the coding/computational thinking standards. The computer literacy benchmarks are perhaps more vague - to effectively use computer input/output devices and conduct research an in age-appropriate way.

Selected CSTA K-2 benchmarks:

Definition 3.3.

CODING

L0CP04 Construct a set of statements to be acted out to accomplish a simple task (e.g., turtle instructions).

L0CT02 Use writing tools, digital cameras, and drawing tools to illustrate thoughts, ideas, and stories in a step-by-step manner.

L0CT03 Understand how to arrange (sort) information into useful order, such as sorting students by birth date, without using a computer.

COMPUTER LITERACY

L0CD01 Use standard input and output devices to successfully operate computers and related technologies.

L0CP01 Use technology resources to conduct age-appropriate research.

In grades 3-6, coding standards include the ability to implement problem solutions using block-based programming like Scratch, to understand and develop simple algorithms, and to break up larger problems into sub-problems. Computer literacy standards echo many subjects that arose during the LS brainstorm: proficiency with word processing, email, keyboarding, troubleshooting, internet safety and ethics, and web searching.

Selected CSTA standards for grades 3-6:

Definition 3.4.

CODING

L1CP05 Construct a program as a set of step-by-step instructions to be acted out.

L1CP06 Implement problem solutions using a blockbased visual programming language.

L1CT01 Understand and use the basic steps in algorithmic problem-solving (e.g., problem statement and exploration, examination of sample instances, design, implementation, and testing).

L1CT02 Develop a simple understanding of an algorithm (e.g., search, sequence of events, or sorting) using computer-free exercises.

L1CT05 Make a list of sub-problems to consider while addressing a larger problem.

COMPUTER LITERACY

L1C01 Use productivity technology tools (e.g., word processing, spreadsheet, presentation software) for individual and collaborative writing, communication, and publishing activities.

L1C02 Use online resources (e.g., email, online discussions, collaborative web environments) to participate in collaborative problemsolving activities for the purpose of developing solutions or products.

L1CD1 Demonstrate an appropriate level of proficiency with keyboards and other input and output devices

L1CD3 Apply strategies for identifying simple hardware and software problems that may occur during use.

L1CI01 Discuss basic issues related to responsible use of technology and information, and the consequences of inappropriate use.

L1CI02 Identify the impact of technology (e.g., social networking, cyber bullying, mobile computing and communication, web technologies, cyber security, and virtualization) on personal life and society.

L1CI03 Evaluate the accuracy, relevance, appropriateness, comprehensiveness, and biases that occur in electronic information sources.

L1CI04 Understand ethical issues that relate to computers and networks (e.g., equity of access, security, privacy, copyright, and intellectual property).

L1CP04 Gather and manipulate data using a variety of digital tools.

L1CP08 Navigate between webpages using hyperlinks and conduct simple searches using search engines.

At this point in time, LS Newman curriculum is on track to meet the CSTA standards. Specific program recommendations are discussed in the next section.

US

As of writing, CS exposure at Newman is largely self-guided and limited to a small group of students. Approximately 5 students are enrolled in Online Global Academy, an online platform offering courses in a variety of subject areas. The CS courses include:

- Art of Code: Processing
- Computer Programming I: Computational Thinking
- Computer Programming I: Java
- Computer Programming II: Advanced Java
- Computer Programming II: Analyzing Data with Python
- Computer Programming II: iOS Development

Students in Tech Club meet on a weekly basis to build and program their autonomous drone. Coding Club ostensibly engaged students in extracurricular CS, although, like many student groups, participation quickly waned as students got bored with online exercises. Several other students cite summer coding programs or online platforms like Kahn Academy for exposure to programming languages like Python and C++. Although a small handful of students are finding time and ways to code, Newman's Upper School is not providing adequate opportunities to develop critical foundational CS skills, nor is it reaching an adequate portion of the student body.

Greg Malis, an US math teacher, argues that computer science fits into both science and math: "What we do in math is explore patterns, algorithms, and there is no more concrete version of an algorithm than code." As for science: "It fits. The ultimate goal of science is to formulate hypotheses, test, and debug. That, in a nutshell, is what coding's about." When asked about the role of computer science in K-12 education, Malis responded: "Our job is to prepare students. If our students graduate without any exposure to coding, we aren't doing our job."

Chapter 4

Recommendations

4.1 Lower School

4.2 Middle School

Possible implementation strategies to explore: CS inegrated in the classroom, 6th grade science course, club time, flex time, new schedule

4.3 Upper School

Unlike the Middle School schedule, the Upper School sequence affords time and space for computer science electives, as well as a graduation requirement. The rotating eight block schedule means there are 32 possible credits, although Newman students rarely graduate with greater than 30, typically averaging approximately 27 credits. The student handbook stipulates that students must complete 24 academic credits to graduate, taken from the following subject areas:

Subject	**Credits**
English	4
Mathematics	4
History/Social Studies	4
Science	3
World Languages	3
Arts	2
Physical Education	2
Electives	2
Total	**24**

Table 4.1: Upper School Graduation Requirements

The "elective" graduation requirement is the most obvious space to insert CS courses. It's worth examining which departments and classes may be impacted by these additional electives.

If the ultimate objective is to treat CS as foundational subject on par with core subjects like math and English, a critical analysis of the Upper School curriculum will be required to make space for a new set of mandated credits. Given the nascent status of computer science at Newman, this document will not attempt to evaluate the merits of the existing sequence or propose potentially subversive structural changes. Nevertheless, it's instructive to examine schools with robust CS requirements and evaluate how a similar program at Newman might affect existing courses.

4.3.1 Standards

4.3.2 AP Computer Science

4.3.3 Online Education

4.3.4 Tiered CS Electives

A scaffolded series of CS courses is the best way to ensure

4.3.5 CS Department

4.3.6 Innovation, Design Thinking, & CS

4.3.7 CS Graduation Requirement

This document strongly recommends that the Upper School adopt a 1/2 credit CS graduation requirement. A mandated credit, as opposed to an optional elective, ensures that every student- especially groups typically underrepresented in CS- get exposure to coding prior to college and have the opportunity to explore coding career possibilities prior to choosing a college major or minor.

In addition to introducing foundational computational/coding concepts (explored in the "Benchmarks" section), this introductory CS course should prioritize the following the exploration diverse career opportunities afforded by CS/computational thinking, as well as emphasize the potential for innovation and creative expression through code. Assuming this 1/2 credit course is the single point of CS exposure, the course needs to serve as the hook. Given the multitude of coding languages and platforms (refer to Appendix A), a highly-engaging project-based course that explores a myriad of hardware and software tools is *both feasible and critically important* for an effective introductory CS experience.

Included in Appendix **??** are suggested syllabi for two 1/2 credit courses– “Creative Coding” and “Physical Computing.” Each course is designed to meet the learning objectives of an introductory series as outlined above and as specified by CSTA Benchmarks.

- staffing requirements

- space requirements

- potential for online, self-guided instruction

Appendix A

Appendix: Online Curriculum

The following table is compiled from EdSurge[38] and Code.Org[?]. For full descritions and side-by-side comparisons, visit Code.org's 3rd Educator Party Resources.

Table A.1: K-12 Online Curriculum

Org	**Curriculum**	**PD**	**Cost**	**Self-Guided?**	**Student Tracking?**
LOWER SCHOOL					
Code Studio (Code.org)	4 courses blend online tutorials with "unplugged" activities	1-day weekend workshops across the US, FREE	FREE	Y	Y
Hopscotch	This free iPad app uses a visual programming language similar to Scratch to help kids learn the basics of programming logic, such as sequencing, loops, variables, functions and conditionals.	-	FREE	N	N
Kodable	Kodable is a freemium educational iPad game offering a kid-friendly introduction to programming concepts and problem solving.	-	FREE	Y	Y
Project Lead The Way	6, 10-hour computer science modules	Face-to-face and online, $700 for school-level lead teacher	$750 /school	Y	Y
Scratch Jr.	An iPad app that introduces coding cocepts through games and visual programming. Assessments and curriculum included.	-	FREE	Y	N

ScratchEd	A 6-unit intro to Scratch, a visual programming language for game creation, animations, and art.	In-person educator meet-ups and online MOOC, FREE	FREE	N	N
SNAP!	SNAP!'s visual blocks (similar to Scratch) support higher level computer science concepts like recursion, procedures, and continuations, SNAP! can work with LEGO Mindstorms NXT.	-	FREE	N	N
Tynker	Inspired by Scratch, Tynker has a dashboard to allow teachers to create a more structured way of teaching code with visual blocks. Includes assessment, classroom management, lesson plans, and a built in tutor.	2-day PD, $2000/day + travel	FREE tutorials. $399 /class, $2,000 /school	Y	Y
MIDDLE SCHOOL					
Bootstrap	Teach algebra through video-game programming, with a 20-hr module to go alongside or inside a math class	3-day workshops for schools and districts. Fees range	FREE	N	N
Code.org	20-lesson CS in Algebra and CS in Science modules for math and science classes	In-person and online workshops available to partner districts, FREE, 50% of teacher stipends covered by Code.org	FREE	Y	Y
CodeHS	Year-long Introduction to Computer Science and AP Computer Science in Java.	Online PD for teaching intro computer science, 30-40 hour course, $1000/teacher	FREE Intro; Pro version is $2000/ section /year	Y	Y
Globaloria	6 game-design courses	3-day, in-person training and ongoing online PD, fee included in student price	$75 /student	Y	Y
NCLab	Courses in Karel coding, Turtle coding, 3D modeling, and Python.	Online	$899 /class	Y	Y

Pluaralsight	Free video-led coding courses for kids on Scratch, HTML, App Inventor, Kodu and Hopscotch	Online	FREE	Y	N
Project Lead The Way	6, 10-hour computer science modules	Face-to-face and online, $700 for school-level lead teacher	$750 /school	Y	Y
ScratchEd	A 6-unit intro to Scratch	In-person educator meet-ups and online MOOC, FREE	FREE	N	N
SNAP!	SNAP!'s visual blocks (similar to Scratch) support higher level computer science concepts like recursion, procedures, and continuations, SNAP! can work with LEGO Mindstorms NXT.	See Scratch	FREE	N	N
Tynker	Inspired by Scratch, Tynker has a dashboard to allow teachers to create a more structured way of teaching code with visual blocks. Includes assessment, classroom management, lesson plans, and a built in tutor.	2-day PD, $2000/day + travel	Free tutorials. $399 /class, $2,000 /school	Y	Y
UPPER SCHOOL					
Beauty and Joy of Computing	Year-long CS Principles course	In-person in NYC, Berkeley, CA and North Carolina, FREE, stipends in NYC, stipends + travel elsewhere paid as available	FREE	N	N
Bootstrap	Teach algebra through video-game programming, with a 20-hr module to go alongside or inside a math class	3-day workshops for schools and districts. Fees range	FREE	N	N

Code Avengers	In-browser exercises and courses in JavaScript, HTML5, CSS3 and Python. Introductory courses are free, with intermediate and advanced courses for 29−39.	Online	FREE intro; intermediate and advanced courses for $29-$39.	Y	Y
Code.org	Exploring Computer Science intro course and Computer Science Principles, a pilot course with an AP exam coming in 2016	In-person and online workshops available to partner districts, FREE, 50% of teacher stipends covered by Code.org	FREE	N	N
Codecademy	6 online courses (Javascript, SQL, Python, Ruby on Rails, Git, HTML/CSS, and more) and 65 projects. FREE	Not provided	FREE	Y	Y
CodeHS	Year-long Introduction to Computer Science and AP Computer Science in Java.	Online PD for teaching intro computer science, 30-40 hour course, $1000/teacher	FREE Intro; Pro version is $2000/section/year	Y	Y
Edhesive	Year-long AP Computer Science course	Online PD, community and content/technical/program support available, $2,200 per school	FREE	Y	Y
Globaloria	6 game-design courses	3-day, in-person training and ongoing online PD, fee included in student price	$75/student	Y	Y
Hello Processing	Video tutorials and interactive coding environment for learning Processing.	Online	FREE	Y	N
Khan Academy	Online curriculum that teaches JavaScript programming, HTML/CSS, and SQL, in an interactive online environment, plus courses on Algorithms and Cryptography.	Online	FREE	Y	Y

Mobile CSP	Year-long AP Computer Science Principles course based on App Inventor, a mobile programming language for Android devices. Materials available online	Online, regional in-person offered in CT, MA, NH and CA (others may be available), FREE, stipends available	FREE	Y	N
NCLab	Courses in Karel coding, Turtle coding, 3D modeling, and Python.	Online	$899 /class	Y	Y
Project Lead The Way	Several courses: Intro CS, AP, Cybersecurity, and CS Applications	5 or 10-day in-person training, $1200 or $2400, depending on course	$2000 /school	Y	Y

Appendix B

Appendix: Hardware Tools

TABLE B.1: Hardware Tools for Teaching Coding

Tool	Language	Description	Ages	Price
Arduino	Arduino (C++), Scratch	Programmable tool for developing interactive electronics projects	10+	$12
Bee-Bots	Bee-Bot	An easy-to-operate robot for computational thinking	4 to 7	$90
LEGO Mindstorms NXT/EV3	Mindstorms	Programmable LEGO robots with a variety of sensors; online curriculum	9+	$350+
LittleBits	Arduino	Easy-to-use electronic building blocks that can be programmed with Arduino	5+	$89
Makey Makey	Arduino, Scratch	A hardware “invention kit” that integrates well with Scratch	7+	$60
Ozobot	Ozobot	A tiny robot that teaches coding through game-based apps	5+	$50
Raspberri Pi	Python	Tiny, affordable computer to learn programming through fun projects	12+	$50
Sphero	Sphero	LED lit, programmable robot connects to iOS and Android	5+	$99
VEX Robotics	VEX	A rich, programmable robotics platform for classroom or competition	12+	$400

Appendix C

Appendix: CSTA Benchmarks

The Computer Science Teachers Association (CSTA) K-12 Computer Science Standards (revised 2011) [28] are broken into tables by levels and grouped by "strand": computational thinking; collaboration; computing practice; computers and communication devices; and community, global, and ethical impacts.

TABLE C.1: K-3rd Grade CS Benchmarks

ID	**DESCRIPTION**
	COLLABORATION
L0C01	Gather information and communicate electronically with others with support from teachers, family members, or student partners.
L0C02	Work cooperatively and collaboratively with peers, teachers, and others using technology
	COMPUTERS AND COMMUNICATION DEVICES
L0CD01	Use standard input and output devices to successfully operate computers and related technologies.
	COMPUTING PRACTICE AND PROGRAMMING
L0CP01	Use technology resources to conduct age-appropriate research.
L0CP02	Use developmentally appropriate multimedia resources (e.g., interactive books and educational software) to support learning across the curriculum.
L0CP03	Create developmentally appropriate multimedia products with support from teachers, family members, or student partners.
L0CP04	Construct a set of statements to be acted out to accomplish a simple task (e.g., turtle instructions).
L0CP05	Identify jobs that use computing and technology.
L0CP06	Gather and organize information using concept-mapping tools.
	COMPUTATIONAL THINKING
L0CT01	Use technology resources (e.g., puzzles, logical thinking programs) to solve ageappropriate problems
L0CT02	Use writing tools, digital cameras, and drawing tools to illustrate thoughts, ideas, and stories in a step-by-step manner.
L0CT03	Understand how to arrange (sort) information into useful order, such as sorting students by birth date, without using a computer.
L0CT04	Recognize that software is created to control computer operations.
L0CT05	Demonstrate how 0s and 1s can be used to represent information.

Table C.2: 3rd-6th Grade CS Benchmarks

ID	**DESCRIPTION**
	COLLABORATION
L1C01	Use productivity technology tools (e.g., word processing, spreadsheet, presentation software) for individual and collaborative writing, communication, and publishing activities.
L1C02	Use online resources (e.g., email, online discussions, collaborative web environments) to participate in collaborative problemsolving activities for the purpose of developing solutions or products.
L1C03	Identify ways that teamwork and collaboration can support problem solving and innovation.
	COMPUTERS AND COMMUNICATION DEVICES
L1CD1	Demonstrate an appropriate level of proficiency with keyboards and other input and output devices
L1CD2	Understand the pervasiveness of computers and computing in daily life (e.g., voice mail, downloading videos and audio files, microwave ovens, thermostats, wireless Internet, mobile computing devices, GPS systems).
L1CD3	Apply strategies for identifying simple hardware and software problems that may occur during use.
L1CD4	Identify that information is coming to the computer from many sources over a network.
L1CD5	Identify factors that distinguish humans from machines.
L1CD6	Recognize that computers model intelligent behavior (as found in robotics, speech and language recognition, and computer animation).
	COMMUNITY, GLOBAL, AND ETHICAL IMPACTS
L1CI01	Discuss basic issues related to responsible use of technology and information, and the consequences of inappropriate use.
L1CI02	Identify the impact of technology (e.g., social networking, cyber bullying, mobile computing and communication, web technologies, cyber security, and virtualization) on personal life and society.
L1CI03	Evaluate the accuracy, relevance, appropriateness, comprehensiveness, and biases that occur in electronic information sources.
L1CI04	Understand ethical issues that relate to computers and networks (e.g., equity of access, security, privacy, copyright, and intellectual property).
	COMPUTING PRACTICE AND PROGRAMMING
L1CP01	Use technology resources (e.g., calculators, data collection probes, mobile devices, videos, educational software, and web tools) for problem-solving and self-directed learning.
L1CP02	Use general-purpose productivity tools and peripherals to support personal productivity, remediate skill deficits, and facilitate learning.
L1CP03	Use technology tools (e.g., multimedia and text authoring, presentation, web tools, digital cameras, and scanners) for individual and collaborative writing, communication, and publishing activities.
L1CP04	Gather and manipulate data using a variety of digital tools.
L1CP05	Construct a program as a set of step-by-step instructions to be acted out (e.g., make a peanut butter and jelly sandwich activity).
L1CP06	Implement problem solutions using a blockbased visual programming language.

L1CP07	Use computing devices to access remote information, communicate with others in support of direct and independent learning, and pursue personal interests.
L1CP08	Navigate between webpages using hyperlinks and conduct simple searches using search engines.
L1CP09	Identify a wide range of jobs that require knowledge or use of computing.
L1CP10	Gather and manipulate data using a variety of digital tools.
	COMPUTATIONAL THINKING
L1CT01	Understand and use the basic steps in algorithmic problem-solving (e.g., problem statement and exploration, examination of sample instances, design, implementation, and testing).
L1CT02	Develop a simple understanding of an algorithm (e.g., search, sequence of events, or sorting) using computer-free exercises.
L1CT03	Demonstrate how a string of bits can be used to represent alphanumeric information.
L1CT04	Describe how a simulation can be used to solve a problem.
L1CT05	Make a list of sub-problems to consider while addressing a larger problem.
L1CT06	Understand the connections between computer science and other fields.

Table C.3: Level 2 CS Benchmarks

ID	**DESCRIPTION**
	COLLABORATION
L2C01	Apply productivity/multimedia tools and peripherals to group collaboration and support learning throughout the curriculum.
L2C02	Collaboratively design, develop, publish, and present products (e.g., videos, podcasts, websites) using technology resources that demonstrate and communicate curriculum concepts.
L2C03	Collaborate with peers, experts, and others using collaborative practices such as pair programming, working in project teams, and participating in group active learning activities.
L2C04	Exhibit dispositions necessary for collaboration: providing useful feedback, integrating feedback, understanding and accepting multiple perspectives, socialization.
	COMPUTERS AND COMMUNICATION DEVICES
L2CD01	Recognize that computers are devices that execute programs.
L2CD02	Identify a variety of electronic devices that contain computational processors.
L2CD03	Demonstrate an understanding of the relationship between hardware and software.
L2CD04	Use developmentally appropriate, accurate terminology when communicating about technology.
L2CD05	Apply strategies for identifying and solving routine hardware problems that occur during everyday computer use.
L2CD06	Describe the major components and functions of computer systems and networks.
L2CD07	Describe what distinguishes humans from machines focusing on human intelligence versus machine intelligence and ways we can communicate.
L2CD08	Describe ways in which computers use models of intelligent behavior (e.g., robot motion, speech and language understanding, and computer vision).
	COMMUNITY, GLOBAL, AND ETHICAL IMPACTS

L2CI01	Exhibit legal and ethical behaviors when using information and technology and discuss the consequences of misuse.
L2CI02	Demonstrate knowledge of changes in information technologies over time and the effects those changes have on education, the workplace, and society.
L2CI03	Analyze the positive and negative impacts of computing on human culture.
L2CI04	Evaluate the accuracy, relevance, appropriateness, comprehensiveness, and bias of electronic information sources concerning real-world problems.
L2CI05	Describe ethical issues that relate to computers and networks (e.g., security, privacy, ownership, and information sharing).
L2CI06	Discuss how the unequal distribution of computing resources in a global economy raises issues of equity, access, and power.
	COMPUTING PRACTICE AND PROGRAMMING
L2CP01	Select appropriate tools and technology resources to accomplish a variety of tasks and solve problems.
L2CP02	Use a variety of multimedia tools and peripherals to support personal productivity and learning throughout the curriculum.
L2CP03	Design, develop, publish, and present products (e.g., webpages, mobile applications, animations) using technology resources that demonstrate and communicate curriculum concepts.
L2CP04	Demonstrate an understanding of algorithms and their practical application.
L2CP05	Implement problem solutions using a programming language, including: looping behavior, conditional statements, logic, expressions, variables, and functions.
L2CP06	Demonstrate good practices in personal information security, using passwords, encryption, and secure transactions.
L2CP07	Identify interdisciplinary careers that are enhanced by computer science.
L2CP08	Demonstrate dispositions amenable to openended problem solving and programming (e.g., comfort with complexity, persistence, brainstorming, adaptability, patience, propensity to tinker, creativity, accepting challenge).
L2CP09	Collect and analyze data that is output from multiple runs of a computer program.
	COMPUTATIONAL THINKING
L2CT01	Use the basic steps in algorithmic problemsolving to design solutions (e.g., problem statement and exploration, examination of sample instances, design, implementing a solution, testing, evaluation).
L2CT02	Describe the process of parallelization as it relates to problem solving.
L2CT03	Define an algorithm as a sequence of instructions that can be processed by a computer.
L2CT04	Evaluate ways that different algorithms may be used to solve the same problem.
L2CT05	Act out searching and sorting algorithms.
L2CT06	Describe and analyze a sequence of instructions being followed (e.g., describe a character's behavior in a video game as driven by rules and algorithms).
L2CT07	Represent data in a variety of ways including text, sounds, pictures, and numbers.
L2CT08	Use visual representations of problem states, structures, and data (e.g., graphs, charts, network diagrams, flowcharts).
L2CT09	Interact with content-specific models and simulations (e.g., ecosystems, epidemics, molecular dynamics) to support learning and research.

L2CT10	Evaluate what kinds of problems can be solved using modeling and simulation.
L2CT11	Analyze the degree to which a computer model accurately represents the real world.
L2CT12	Use abstraction to decompose a problem into sub problems.
L2CT13	Understand the notion of hierarchy and abstraction in computing including highlevel languages, translation, instruction set, and logic circuits.
L2CT14	Examine connections between elements of mathematics and computer science including binary numbers, logic, sets and functions.
L2CT15	Provide examples of interdisciplinary applications of computational thinking.

Table C.4: Level 3A CS Benchmarks

ID	**DESCRIPTION**
	COLLABORATION
L3AC01	Work in a team to design and develop a software artifact.
L3AC02	Use collaborative tools to communicate with project team members (e.g., discussion threads, wikis, blogs, version control, etc.).
L3AC03	Describe how computing enhances traditional forms and enables new forms of experience, expression, communication, and collaboration
L3AC04	Identify how collaboration influences the design and development of software products.
	COMPUTERS AND COMMUNICATION DEVICES
L3ACD01	Describe the unique features of computers embedded in mobile devices and vehicles (e.g., cell phones, automobiles, airplanes).
L3ACD02	Develop criteria for purchasing or upgrading computer system hardware.
L3ACD03	Describe the principal components of computer organization (e.g., input, output, processing, and storage).
L3ACD04	Compare various forms of input and output.
L3ACD05	Explain the multiple levels of hardware and software that support program execution (e.g., compilers, interpreters, operating systems, networks).
L3ACD06	Apply strategies for identifying and solving routine hardware and software problems that occur in everyday life.
L3ACD07	Compare and contrast client-server and peer-to-peer network strategies.
L3ACD08	Explain the basic components of computer networks (e.g., servers, file protection, routing, spoolers and queues, shared resources, and fault-tolerance).
L3ACD09	Describe how the Internet facilitates global communication.
L3ACD10	Describe the major applications of artificial intelligence and robotics.
	COMMUNITY, GLOBAL, AND ETHICAL IMPACTS
L3ACI01	Compare appropriate and inappropriate social networking behaviors.
L3ACI02	Discuss the impact of computing technology on business and commerce (e.g., automated tracking of goods, automated financial transactions, e-commerce, cloud computing).
L3ACI03	Describe the role that adaptive technology can play in the lives of people with special needs.
L3ACI04	Compare the positive and negative impacts of technology on culture (e.g., social networking, delivery of news and other public media, and intercultural communication).
L3ACI05	Describe strategies for determining the reliability of information found on the Internet.

L3ACI06	Differentiate between information access and information distribution rights.
L3ACI07	Describe how different kinds of software licenses can be used to share and protect intellectual property.
L3ACI08	Discuss the social and economic implications associated with hacking and software piracy.
L3ACI09	Describe different ways in which software is created and shared and their benefits and drawbacks (commercial software, public domain software, open source development).
L3ACI10	Describe security and privacy issues that relate to computer networks.
L3ACI11	Explain the impact of the digital divide on access to critical information.
	COMPUTING PRACTICE AND PROGRAMMING
L3ACP01	Create and organize Web pages through the use of a variety of web programming design tools.
L3ACP02	Use mobile devices/emulators to design, develop, and implement mobile computing applications.
L3ACP03	Use various debugging and testing methods to ensure program correctness (e.g., test cases, unit testing, white box, black box, integration testing)
L3ACP04	Apply analysis, design, and implementation techniques to solve problems (e.g., use one or more software lifecycle models).
L3ACP05	Use Application Program Interfaces (APIs) and libraries to facilitate programming solutions.
L3ACP06	Select appropriate file formats for various types and uses of data.
L3ACP07	Describe a variety of programming languages available to solve problems and develop systems.
L3ACP08	Explain the program execution process.
L3ACP09	Explain the principles of security by examining encryption, cryptography, and authentication techniques.
L3ACP10	Explore a variety of careers to which computing is central.
L3ACP11	Describe techniques for locating and collecting small and large-scale data sets.
L3ACP12	Describe how mathematical and statistical functions, sets, and logic are used in computation.
	COMPUTATIONAL THINKING
L3ACT01	Use predefined functions and parameters, classes and methods to divide a complex problem into simpler parts.
L3ACT02	Describe a software development process used to solve software problems (e.g., design, coding, testing, verification).
L3ACT03	Explain how sequence, selection, iteration, and recursion are building blocks of algorithms.
L3ACT04	Compare techniques for analyzing massive data collections.
L3ACT05	Describe the relationship between binary and hexadecimal representations.
L3ACT06	Analyze the representation and trade-offs among various forms of digital information.
L3ACT07	Describe how various types of data are stored in a computer system.
L3ACT08	Use modeling and simulation to represent and understand natural phenomena.
L3ACT09	Discuss the value of abstraction to manage problem complexity.
L3ACT10	Describe the concept of parallel processing as a strategy to solve large problems.
L3ACT11	Describe how computation shares features with art and music by translating human intention into an artifact.

Table C.5: Level 3B CS Benchmarks

ID	**DESCRIPTION**
	COLLABORATION
L3BC01	Use project collaboration tools, version control systems, and Integrated Development Environments (IDEs) while working on a collaborative software project.
L3BC02	Demonstrate the software life cycle process by participating on a software project team.
L3BC03	Evaluate programs written by others for readability and usability
	COMPUTERS AND COMMUNICATION DEVICES
L3BCD01	Discuss the impact of modifications on the functionality of application programs.
L3BCD02	Identify and describe hardware (e.g., physical layers, logic gates, chips, components).
L3BCD03	Identify and select the most appropriate file format based on trade-offs (e.g., accuracy, speed, ease of manipulation).
L3BCD04	Describe the issues that impact network functionality (e.g., latency, bandwidth, firewalls, server capability).
L3BCD05	Explain the notion of intelligent behavior through computer modeling and robotics.
	COMMUNITY, GLOBAL, AND ETHICAL IMPACTS
L3BCI01	Demonstrate ethical use of modern communication media and devices.
L3BCI02	Analyze the beneficial and harmful effects of computing innovations.
L3BCI03	Summarize how financial markets, transactions, and predictions have been transformed by automation.
L3BCI04	Summarize how computation has revolutionized the way people build real and virtual organizations and infrastructures.
L3BCI05	Identify laws and regulations that impact the development and use of software.
L3BCI06	Analyze the impact of government regulation on privacy and security.
L3BCI07	Differentiate among open source, freeware, and proprietary software licenses and their applicability to different types of software.
L3BCI08	Relate issues of equity, access, and power to the distribution of computing resources in a global society.
	COMPUTING PRACTICE AND PROGRAMMING
L3BCP01	Use advanced tools to create digital artifacts (e.g., web design, animation, video, multimedia)
L3BCP02	Use tools of abstraction to decompose a large-scale computational problem (e.g., procedural abstraction, object-oriented design, functional design).
L3BCP03	Classify programming languages based on their level and application domain
L3BCP04	Explore principles of system design in scaling, efficiency, and security.
L3BCP05	Deploy principles of security by implementing encryption and authentication strategies.
L3BCP06	Anticipate future careers and the technologies that will exist.
L3BCP07	Use data analysis to enhance understanding of complex natural and human systems.
L3BCP08	Deploy various data collection techniques for different types of problems.
	COMPUTATIONAL THINKING
L3BCT01	Classify problems as tractable, intractable, or computationally unsolvable.

L3BCT02	Explain the value of heuristic algorithms to approximate solutions for intractable problems.
L3BCT03	Critically examine classical algorithms and implement an original algorithm.
L3BCT04	Evaluate algorithms by their efficiency, correctness, and clarity.
L3BCT05	Use data analysis to enhance understanding of complex natural and human systems.
L3BCT06	Compare and contrast simple data structures and their uses (e.g., arrays and lists).
L3BCT07	Discuss the interpretation of binary sequences in a variety of forms (e.g., instructions, numbers, text, sound, image).
L3BCT08	Use models and simulations to help formulate, refine, and test scientific hypotheses.
L3BCT09	Analyze data and identify patterns through modeling and simulation.
L3BCT10	Decompose a problem by defining new functions and classes.
L3BCT11	Demonstrate concurrency by separating processes into threads and dividing data into parallel streams.

Appendix D

Appendix: ISTE Standards

The International Society for Technology in Education (ISTE) compiled the "Computer Science Standards for Educators" [29]. Where applicable, the ISTE CS standards are linked to CSTA standards.

1. Demonstrate knowledge of Computer Science content and model important principles and concepts.
   (a) Demonstrate knowledge of and proficiency in data representation and abstraction L2CT12, L3BCP02
      i. Effectively use primitive data types L3ACT07
      ii. Demonstrate an understanding of static and dynamic data structures L3BCT06
      iii. Effectively use, manipulate, and explain various external data stores: various types (text, images, sound, etc.), various locations (local, server, cloud), etc. L2CT07, L3ACP06, L3ACT07
      iv. Effectively use modeling and simulation to solve real-world problems L3ACP04, L3ACT08, L3BCT08, L3BCT09
   (b) Effectively design, develop, and test algorithms
      i. Using a modern, high-level programming language, construct correctly functioning programs involving simple and structured data types; compound boolean expressions; and sequential, conditional, and iterative control structures L2CP05
      ii. Design and test algorithms and programming solutions to problems in different contexts (textual, numeric, graphic, etc.) using advanced data structures L2CT01
      iii. Analyze algorithms by considering complexity, efficiency, aesthetics, and correctness L3BCT04
      iv. Demonstrate knowledge of two or more programming paradigms L3ACP07
      v. Effectively use two or more development environments L3BC01
      vi. Demonstrate knowledge of varied software development models and project management strategies L3ACT02
   (c) Demonstrate knowledge of digital devices, systems, and networks
      i. Demonstrate an understanding of data representation at the machine level L3BCT07
      ii. Demonstrate an understanding of machinelevel components and related issues of complexity L3BCD02
      iii. Demonstrate an understanding of operating systems and networking in a structured computer system L3ACD05
      iv. Demonstrate an understanding of the operation of computer networks and mobile computing devices L3ACD01

(d) Demonstrate an understanding of the role computer science plays and its impact in the modern world L3BCI02

i. Demonstrate an understanding of the social, ethical, and legal issues and impacts of computing, and attendant responsibilities of computer scientists and users L3ACI04

ii. Analyze the contributions of computer science to current and future innovations in sciences, humanities, the arts, and commerce L3BCI04

2. Effective teaching and learning strategies

   (a) Plan and teach computer science lessons/units using effective and engaging practices and methodologies

   i. Select a variety of real-world computing problems and project-based methodologies that support active and authentic learning and provide opportunities for creative and innovative thinking and problem solving

   ii. Demonstrate the use of a variety of collaborative groupings in lesson plans/units and assessments

   iii. Design activities that require students to effectively describe computing artifacts and communicate results using multiple forms of media

   iv. Develop lessons and methods that engage and empower learners from diverse cultural and linguistic backgrounds

   v. Identify problematic concepts and constructs in computer science and appropriate strategies to address them

   vi. Design and implement developmentally appropriate learning opportunities supporting the diverse needs of all learners

   vii. Create and implement multiple forms of assessment and use resulting data to capture student learning, provide remediation, and shape classroom instruction

3. Effective, safe, & ethical learning environments

   (a) Design environments that promote effective teaching and learning in computer science classrooms and online learning environments and promote digital citizenship

   i. Promote and model the safe and effective use of computer hardware, software, peripherals, and networks

   ii. Plan for equitable and accessible classroom, lab, and online environments that support effective and engaging learning

4. Effective professional knowledge and skills

   (a) Participate in, promote, and model ongoing professional development and life-long learning relative to computer science and computer science education

   i. Identify and participate in professional computer science and computer science education societies, organizations, and groups that provide professional growth opportunities and resources

   ii. Demonstrate knowledge of evolving social and research issues relating to computer science and computer science education

   iii. Identify local, state, and national content and professional standards and requirements affecting the teaching of secondary computer science

Appendix E

Appendix: NGSS Standards

Table E.1: K-12 Engineering Design NGSS Standards [39]

ID	DESCRIPTION
K-2-ETS1-1.	Ask questions, make observations, and gather information about a situation people want to change to define a simple problem that can be solved through the development of a new or improved object or tool.
K-2-ETS1-2.	Develop a simple sketch, drawing, or physical model to illustrate how the shape of an object helps it function as needed to solve a given problem.
K-2-ETS1-3.	Analyze data from tests of two objects designed to solve the same problem to compare the strengths and weaknesses of how each performs.
3-5-ETS1-1.	Define a simple design problem reflecting a need or a want that includes specified criteria for success and constraints on materials, time, or cost.
3-5-ETS1-2.	Generate and compare multiple possible solutions to a problem based on how well each is likely to meet the criteria and constraints of the problem.
3-5-ETS1-3.	Plan and carry out fair tests in which variables are controlled and failure points are considered to identify aspects of a model or prototype that can be improved.
MS-ETS1-1.	Define the criteria and constraints of a design problem with sufficient precision to ensure a successful solution, taking into account relevant scientific principles and potential impacts on people and the natural environment that may limit possible solutions.
MS-ETS1-2.	Evaluate competing design solutions using a systematic process to determine how well they meet the criteria and constraints of the problem.
MS-ETS1-3.	Analyze data from tests to determine similarities and differences among several design solutions to identify the best characteristics of each that can be combined into a new solution to better meet the criteria for success.
MS-ETS1-4.	Develop a model to generate data for iterative testing and modification of a proposed object, tool, or process such that an optimal design can be achieved.
HS-ETS1-1.	Analyze a major global challenge to specify qualitative and quantitative criteria and constraints for solutions that account for societal needs and wants.
HS-ETS1-2.	Design a solution to a complex real-world problem by breaking it down into smaller, more manageable problems that can be solved through engineering.

HS-ETS1-3.	Evaluate a solution to a complex real-world problem based on prioritized criteria and trade-offs that account for a range of constraints, including cost, safety, reliability, and aesthetics as well as possible social, cultural, and environmental impacts.
HS-ETS1-4.	Use a computer simulation to model the impact of proposed solutions to a complex real-world problem with numerous criteria and constraints on interactions within and between systems relevant to the problem.

Appendix F

Appendix: Introductory CS Syllabi

Creative Coding, *Fall*
Creative coding uses computer programming as a medium for creative expression. In this course students will use languages like Processing, a visual programming language designed for artists, to code animations or build interactive visualizations on the web. This project-based class also explores the use of programmable tools like Arduinos and 3D printers in the creation of wearable technology and interactive electronic art.

Skills
D3/Javascript (Web Visualization)
Processing/JAVA (Animations)
Arduino/C++ (Wearables)

Physical Computing, *Spring*
Physical computing uses software and hardware to develop interactive physical systems. This course introduces students to coding, robotics, laser cutting, CNC milling, 3D printing, and a variety of programmable tools capable of sensing and responding to real world stimuli. Line-following robots, electronic music instruments, and LED smart bands are just a few examples of potential projects created in this hands-on course.

Skills
Arduino/C++ (Electronics)
Raspberry Pi/Python (Robotics)
OpenSCAD/C++ (3D Printing)

Bibliography

[1] K. Prottsman. Coding vs. programming – battle of the terms! *Huffington Post*, April 2015. URL http://www.huffingtonpost.com/kiki-prottsman/coding-vs-programming-bat_b_7042816.html.

[2] Quinn Burke Yasmin B. Kafai. Computer programming goes back to school. *The Phi Delta Kappan*, 95(1): 61–65, 2013. ISSN 00317217. URL http://www.jstor.org/stable/23617761.

[3] Carnegie Mellon Center for Computational Thinking. What is computational thinking? URL http://www.cs.cmu.edu/~CompThink/.

[4] Google for Education. Exploring computational thinking. URL https://www.google.com/edu/resources/programs/exploring-computational-thinking/.

[5] Moving beyond computer literacy: Why schools should teach computer science. *National Center for Women and Information Technology.* URL https://www.ncwit.org/resources/moving-beyond-computer-literacy-why-schools-should-teach-computer-science/moving-beyond.

[6] Mayor's Press Office. Mayor emanuel announces major gains in computer science 4 all program. *City of Chicago*, December 2014. URL http://www.cityofchicago.org/city/en/depts/mayor/press_room/press_releases/2014/dec/mayor-emanuel-announces-major-gains-in-computer-science-4-all-pr.html.

[7] Y. Koh. New education bill to get more coding in classrooms. *Wall Street Journal*, December 2015. URL http://blogs.wsj.com/digits/2015/12/10/new-education-bill-to-get-more-coding-in-classrooms/.

[8] Stuart Fox. Sophisticated new computer models predict details of insurgent attacks. *Popular Science*, December 2009. URL http://www.popsci.com/technology/article/2009-12/competing-computer-models-predict-insurgent-attacks.

[9] S. Papert. The children's machine: Rethinking school in the age of the computer. *New York: Basic Books*, 1993.

[10] Mitchel Resnick. Technologies for lifelong kindergarten. *Educational Technology Research and Development*, 46, 1998.

[11] Mitchel Resnick & David Siegel. A different approach to coding: How kids are making and remaking themselves from scratch. November 2015. URL https://medium.com/bright/a-different-approach-to-coding-d679b06d83a#.fjqzcpiyj.

[12] Mitchel Resnick. Rethinking learning in the digital age. 2002. URL https://llk.media.mit.edu/papers/mres-wef.pdf.

[13] Employment projections. *Bureau of Labor Statistics*, 2014. URL http://www.bls.gov/emp/tables.htm.

[14] Code.org. Summary of source data for code.org infographics and stats. 2015. URL https://docs.google.com/document/d/1gySkItxiJn_vwb8HIIKNXqen184mRtzDX12cux0ZgZk/pub.

[15] S. Adams. 25 college diplomas with the highest pay. *Forbes*, April 2013. URL http://www.forbes.com/pictures/mkl45kkeg/1-carnegie-mellon-school-of-computer-science/.

[16] Gallup. Searching for computer science: Access and barriers in u.s. k-12 education. 2014. URL http://csedu.gallup.com/.

[17] College Board. Ap® students in college: An analysis of five-year academic careers. 2007. URL http://research.collegeboard.org/sites/default/files/publications/2012/7/researchreport-2007-4-ap-students-college-analysis-five-year-academic-careers.pdf.

[18] Independent School Data Exchange. Index mission statement. URL http://www.indexgroups.org/about/.

[19] Best stem high schools. *U.S. News Education*, 2015. URL http://www.usnews.com/education/best-high-schools/national-rankings/stem.

[20] Thomas Jefferson HS for Science and Technology. Computer science at thomas jefferson. URL https://www.tjhsst.edu/research-academics/math-cs/computer-science/index.html.

[21] Stuyvesant High School. Mathematics. . URL http://stuy.enschool.org/apps/pages/index.jsp?uREC_ID=126659&type=d&termREC_ID=&pREC_ID=253269.

[22] High Technology High School. Hths courses. . URL http://www.hths.mcvsd.org/downloads.

[23] Edison academy: Electronics and computer engineering technology (ecet) program. 2015. URL http://www.mcvts.net/Page/2080.

[24] Westside High School Magnet for Integrated Technology. Computing science and engineering. URL http://www.houstonisd.org/Page/73698.

[25] MIT Office of the Registrar. Course 6: Electrical engineering and computer science iap/spring 2016. URL http://student.mit.edu/catalog/m6a.html.

[26] Pomona College Computer Science Department. Course catalog - computer science. . URL http://catalog.pomona.edu/preview_entity.php?catoid=18&ent_oid=1072&returnto=3635.

[27] Pomona College Computer Science Department. Learning objectives. . URL https://www.pomona.edu/academics/departments/computer-science/courses-requirements/learning-objectives.

[28] Computer Science Teachers Association (CSTA). K-12 computer science standards. 2011. URL http://www.csta.acm.org/Curriculum/sub/CurrFiles/CSTA_K-12_CSS.pdf.

[29] International Society for Technology Education (ISTE). URL http://www.iste.org/standards/iste-standards/standards-for-computer-science-educators.

[30] College Board. Ap computer science principles curriculum framework. *Advances in AP*, December 2015. URL https://advancesinap.collegeboard.org/stem/computer-science-principles/course-details.

[31] John Tierney. Ap classes are a scam. *Atlantic*, October 2012. URL http://www.theatlantic.com/national/archive/2012/10/ap-classes-are-a-scam/263456/.

[32] Philip Guo. Python is now the most popular introductory teaching language at top u.s. universities. *Communications of the ACM*, July 2014. URL http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext.

[33] Philip Guo. Why python is a great language for teaching beginners in introductory programming classes. May 2007. URL http://pgbovine.net/python-teaching.htm.

[34] College Board Advance Placement. Computer science principles faqs. URL https://advancesinap.collegeboard.org/stem/computer-science-principles/faq.

[35] College Board. Ap computer science principles curriculum framework. URL https://secure-media.collegeboard.org/digitalServices/pdf/ap/ap-computer-science-principles-curriculum-framework.pdf.

[36] Next Generation Science Standards. The need for new science standards. 2015. URL http://www.nextgenscience.org/overview-0.

[37] Tynker. Scope and sequence. URL https://www.tynker.com/school/courses/.

[38] K. Bell. A beginner's guide to bringing coding into the classroom. *edSurge*, November 2015. URL https://www.edsurge.com/news/2015-11-30-a-beginner-s-guide-to-bringing-coding-into-the-classroom.

[39] Next Generation Science Standards. Engineering design standards. 2015. URL http://www.nextgenscience.org/search-standards-dci?field_idea_tid%5B%5D=113.