Isidore Newman School

# Computer Science at Newman

*A comprehensive investigation &*
*implementation strategy guide*
*for K-12 computer science*

*Jenna* deBoisblanc

May 2016

# Acknowledgments

I would like to thank Chris Bigenho, the Director of Educational Technology at Greehill School, for his incredible support and willingness to share CS curriculum and benchmarks. Brad Philipson, the Assistant Head of School at Fort Worth Country Day (FWCD), and Aaron Cadle, a upper school CS teacher at FWCD, were extremely gracious and generous when we toured their school this year to observe their CS program.

Xiao Xiao, a Newman alum from ‘06 and a current Ph.D. student in MIT’s Media Lab, helped me finish thinking through the proposed computer science course ecosystem. I am extremely grateful for her input.

I would also like to thank Susie Toso for being such a kind host and brave computer science leader in the lower school! She has been extremely accommodating of my requests to observe coding lessons. Minnette Patton, the Lower School Assistant Head and former computer teacher, has also been extremely helpful in both providing input about CS as well as providing historical context about coding at Newman.

And finally, and most importantly, I have to thank Oskie Creech, without whom this document would not be possible. Oskie has been the most incredible mentor- both as it relates to computer science and to my experiences at Newman in general. He has facilitated faculty meeting brainstorms, offered insight into research, set up meetings with key teachers and administrators, and shared lessons about effective pedagogy. Oskie truly deserves enormous credit for helping to pull this research document together.

# 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 pleasure* because coding is an incredibly versatile 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 pursuing. 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. 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 privileged 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 progenitor 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 interchangeably throughout this document to refer to a set of K-12 curriculum encompassing all three, it 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 Succeeds 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.

### 1.2.1 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 is 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 [8]. Just as reading comprehension and writing composition—processes of solidifying cogent thoughts—form the *true* basis of literacy, so do 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” [8].

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[9].

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?”

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 of coding mechanics or “computer literacy.” It is the ability to think computationally and algorithmically, to develop and model systems, to decompose and debug complex problems, and to explore and verify hypotheses rapidly and virtually. For these reasons, computer science is paramount in education.

### 1.2.2 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 constraints 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 [10]. Computers can be viewed as a "universal construction material" granting rich opportunities for teaching or learning a diversity of subjects [11].

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" [12].

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 [13].

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

### 1.2.3 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. [14]. By 2022 the Bureau of Labor Statistics estimates there will be over 1 million open computing jobs [14]. 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 computing skills in the next decade [15].

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 [16].

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 [15]. 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 [14].

### 1.2.4 Education Gap

Education, especially K-12, has a significant role to play in the growing shortage of computer science graduates. Only 1 in 4 U.S. schools offer computer science classes with computer programming [17], and in 23 states computer science cannot be used to meet math or science graduation requirements [7]. According to the Gallup study [17], 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

| **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 [15]

required by testing, as well as the inability to hire computer science teachers due to availability and budget constraints [17].

### 1.2.5 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 [17].

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 [18].

### 1.2.6 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 transcend scientific disciplines.

**No matter what field of study students choose to pursue, 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

This section examines a variety of schools—traditional and non-traditional, K-12 and post-secondary—to examine CS curriculum, benchmarks, pedagogy, course sequence, tools, and technologies. Independent schools with existing CS programs illustrate a path forward that may be closely aligned to Newman's own academic mission. Examining the top STEM public high schools (as ranked by the U.S. News in 2015), offers a vision of rigorous CS and engineering integration. The CS programs of both engineering and liberal arts colleges and universities sheds light on post-secondary computer science, while an examination of code schools offers insight into an alternate approach to CS education. Finally, a review of local coding initiatives explains the state of coding in the region. Taken together, these examples place Newman's CS program in context both locally and nationally.

### 2.1.1 Independent Schools

#### 2.1.1.1. INDEX Schools

INDEX Schools is a collaboration of over 100 independent schools "to share data, analysis, research, and information to aid member schools in decision-making, policy development, and strategic planning" [19]. In early fall of 2016, nine schools responded to an INDEX listserv query requesting 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: http://bit.ly/1WFdwXN.

Across the board, schools cited ISTE, CSTA, and AP benchmarks. Another salient trend is the use of *physical computing*, the process of building programmable and interactive physical systems (e.g. robotics, Arduinos, Makey Makeys, etc.), in all divisions to make coding hands-on.

Table 2.1: INDEX Upper School Summary

| **School** | **Graduation Req.** | **AP** | **Physical Comp.** | **Tiers** |
|---|---|---|---|---|
| Fort Worth Country Day | | x | x | 4 |
| Greenhill | x | x | x | 4 |
| Lovett | | x | x | 4 |
| Park Tudor | | x | | 4 |
| Pingry | | x | | 3 |
| Porter-Gaud | x | | x | 4 |
| Ravenscroft | | x | | 3 |
| St. Margaret's | | x | | 4 |
| Winchester Thurston | x | x | x | 4 |
| **TOTAL** | 3/9 | 8/9 | 5/9 | (4) 7/9 |

The INDEX upper school programs are summarized in Table 2.1. Almost all of the nine schools (8/9) who completed the survey offer the AP CS A course, although many schools are excited about the new AP CS Principles. All schools offered at least three tiers of CS, with most (7/9) offering 4 tiers. Three upper schools have graduation requirements, although several CS teachers hope to move towards a required credit. Greenhill has a one trimester requirement, Winchester Thurston requires two trimesters, and Porter-Gaud requires one semester.

A typical or traditional approach to CS (refer to Pingry, Park Tudor, and Ravenscroft) begins with an introductory CS course (typically in Java), followed by the AP, and concluding with advanced data structures or algorithms. Many schools, however, are beginning to adopt a much more creative approach to computer science that has broad applications to a myriad of disciplines. The course sequences of the four schools with the most innovative, thorough, and creative programs are listed in Table 2.2. Winchester Thurston, as an example, offers an introductory CS series for humanities, art and music, or math/science, emphasizing the diverse applications of coding. All four of the schools summarized in Table 2.2 offer hands-on physical computing, in the form of robotics or XBox Kinect 3D body simulations.

A summary of the middle school programs can be found in Table 2.3. Seven of the nine schools interviewed offer CS in some form; all but one of these seven schools have a middle school CS requirement that is met either through integration into existing curriculum or through standalone elective courses. Computer literacy is not formally taught or itemized by the these INDEX schools. Like the upper school sequence, physical computing is a common theme; LEGO robotics and Arduinos are used in six of the nine programs that responded to the survey.

At the middle school level integration into existing curriculum is more prevalent than at the upper school level. At Lovett School in Atlanta, Georgia, integration occurs in 6th science through robotics, Scratch, and Arduino programming; in 7th and 8th grade, CS is offered as an elective. At Pingry School in New Jersey, all sixth graders take a co-curricular robotics class, and in 7th and 8th grade, a dedicated CS teacher integrates CS lessons into other classes.

Table 2.2: Select Group of INDEX Upper School CS Programs

| **School** | **Program Summary** |
|---|---|
| Greenhill | 3 tiers, 2 year sequence<br>CS1: Introduction to CS with Arduino / Beginning JAVA Programming / Introduction to Game Design / Engineering<br>CS2: Advanced Computational Design / Intermediate JAVA Programming<br>CS3: Advanced Topics in Computer Science |
| Lovett | Digital Media & Interactive Design - animation, storytelling, 3D & game design (1 semester)<br>CSI - Foundations (1 semester)<br>CSII - Media Computation & App Development (1 semester)<br>CSII - Robotics (1 semester)<br>APCS A - Java Programming (yearlong)<br>Honors Computer Science Studies (yearlong) |
| Porter-Gaud | 9th: Python Game Design, AppInventor App Design, Scribbler Robotics<br>10th: Web Design (HTML,CSS,PHP,SQL) & C#-XNA XBox Game Design<br>11th: Kinect 3D body simulations with C#-XNA<br>12th: Java and iOS Swift |
| Winchester Thurston | 9 courses, 4 tiers<br>Level 1 - (1 trimester) CS for Humanities, CS for Art and Music, CS for Math/Science<br>Level 2 - AP CS, Software Structures, Physical Computing<br>Level 3 - CS Innovations (community involvement)<br>Level 4 - CS Innovations 2, Algorithm Design |

Table 2.3: INDEX Middle School CS Programs

| **School** | **Integrated** | **Art/ Elective** | **Physical Comp.** | **Required** | **Years Required** |
|---|---|---|---|---|---|
| FW Country Day | | | | | 0 |
| Greenhill | | x | x | x | 0.33 |
| Lovett | x | x | x | x | 0.25 |
| Park Tudor | | x | | | 0 |
| Pingry | x | | x | x | 0.25 |
| Porter-Gaud | | x | x | x | 0.5 |
| Ravenscroft | | | | | 0 |
| St. Margaret's | | x | x | x | 0.5 |
| Winchester Thurston | x | x | x | x | 1 |
| **TOTAL** | 3/9 | 6/9 | 6/9 | 6/9 | 0.3(ave) |

Most of the schools interviewed in the survey (6/9) offer standalone CS courses as art electives. At Greenhill School in Dallas, Texas, all 5th and 6th grade students take “Exploratory Design,” a design thinking and engineering CS course that lasts one trimester and shares a elective block with another art credit (refer to the Greenhill section below for additional information). In 7th and 8th grade, CS courses at Greenhill are offered as fine arts electives.

† At St. Margaret’s in San Juan Capistrano, CA, 6th and 7th graders take a quarter long required CS elective. At St. Margaret’s there are seven blocks, five of which are core, one PE, and one elective; the elective is comprised of 1/4 art, 1/4 music, 1/4 health & wellness and 1/4 CS. The CS courses explore LEGO robotics, Arduinos, missions in Mindcraft, and Lua (through Codea). The 8th grade courses is a non-required semester-long elective in wearables, digital fabrication, programming, or art theater.

At Winchester Thurston in Pittsburgh, PA, there is a required, one-trimester elective each year of middle school, in addition to a maker elective. In 6th grade, the “Create Year,” students use Processing to draw pictures; in 7th, the “Interact Year,” students learn user interactions; and in 8th grade, “Application Year,” students use Arduinos to tie CS to other disciplines.

Of the middle school programs that responded to the INDEX query, the average CS requirement is equivalent to one trimester of CS throughout the middle school sequence. Winchester Thurston has the greatest requirement (one trimester each year, equating to one year throughout MS); St. Margaret’s and Porter Gaud have two, quarter-long requirements; and Greenhill, Lovett, and Pingry have the equivalent of one quarter/ trimester CS course.

The lower school programs are summarized in Table 2.4. One of the major trends across divisions and especially in primary education is the use of physical computing hardware (Bee-Bots, LEGO Robotics, Sphero, Makey Makey, etc.) to teach coding. At St. Margaret’s students participate once a week in Imagine, Create, Engineer (ICE), hands-on exploration of design and innovation. Students in K-1 use Bee-Bots, programmable robots, to learn sequencing, logic, and math. Students also learn Scratch, and they develop these skills in 2nd - 5th grades.

One of the primary questions that lower schools face is whether CS should be integrated into existing curriculum or taught as a standalone computer class. Although platforms like Scratch can be used in concert with any subject or curriculum, only two of the nine INDEX schools interviewed in this report integrate coding into other classes. The other major question facing lower school CS teachers is the role of coding vs. computer literacy. Although Greenhill currently blends computer science and computer literacy, Chris Bigenho, the Technology Director, believes that computer literacy should be left to classroom teachers. Mandy Lofquist at Fort Worth Country Day used to have a list of computer literacy benchmarks but has abandoned this list to make way for coding and making.

Table 2.4: INDEX Lower School CS Programs

| School | Physical Comp. | Comp. Literacy | Integration | Standalone | Frequency |
|---|---|---|---|---|---|
| FW Country Day | x | none | | x | K-4: 1/week |
| Greenhill | x | x | | x | 2-4: 1/week |
| Lovett | x | | x | | K-5: Integr. |
| Park Tudor | x | | | | |
| Pingry | x | x | | x | K-5: 1/week |
| Porter-Gaud | | | | | |
| Ravenscroft | x | | | | none |
| St. Margaret's | x | x | | x | K-5: 1/week |
| Winchester Thurston | x | | x | | K-5: Integr. |
| **TOTAL** | 8/9 | 3/9 | 2/9 | 4/9 | |

#### 2.1.1.2. 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, 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 (five minutes) 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.

#### 2.1.1.3. Greenhill School

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, specifically CSTA, 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 they 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 requires two trimesters of "Exploratory Design," an elective that blends 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)

Students may select additional maker or coding courses in 7th and 8th grade. Prototyping I is an optional 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. Second grade students work on word processing, internet research, and educational computer games and apps. Second graders are introduced to coding with MicroWorlds EX. In addition to the skills and topics covered in second grade, third graders learn keyboarding and complete a six week coding project with MicroWorlds. In fourth grade students continue to explore previous subjects in computer literacy and coding in addition to learning file management and emailing.

#### 2.1.1.4. Nueva School

The Nueva School is an independent, PK-12 school in Hillsborough and San Mateo, California, with an enrollment of over 700 students. Nueva is internationally-recognized "for its distinctive inquiry-based interdisciplinary studies, constructivist project-based learning, and its pioneering work in social emotional learning and design thinking" [20]. Design thinking, a human-centered and empathy-driven approach to solving problems, and engineering are heavily emphasized throughout and across divisions. IDEO and Stanford's d.School helped to develop the first PK-12 Design Thinking program and Innovation Lab (I-Lab).

The computer science program at Nueva is only two years old, but because design thinking and engineering are a core philosophy, Nueva's approach to computer science is unique. They currently don't align to any particular curriculum and believe the CSTA is too broad.

In the upper school, CS is used as a problem-solving tool, rather than merely a standalone course, and is integrated into the mathematics and science curriculum. The mathematics department offers a course, "The Elegant Logic of Computer Science," and beginning in grade 9, "students grapple with age-old proofs, unlocking the power of computational programming on today's most robust platform to compile a digital portfolio demonstrating levels of mastery" [21]. Students use Mathematica, a math-based programming language useful for solving integrals or plotting functions. Courses emphasize, among other things, computational thinking and "integrated use of information technology" [21].

Another noteworthy upper school course, "Technology & Design Engineering," combines tools of a makerspace with engineering and design principles:

> "A design innovation class, Design Thinking, Engineering & Entrepreneurship empowers students to further build upon their skills developed in middle school to be change-makers in the world, helping them prepare for a future where resilience is key and translating ideas into reality, essential. Identifying needs and testing them, students generate ideas and create quick prototypes to get feedback, inventing business models to leverage their ideas in the world.

> Use of woodworking tools, CAD software, and digital fabrication equipment, such as the laser cutter and the 3-D printer, helps them bring their ideas to fruition" [21].

Computer science is a tool for innovation and would strongly complement a course of this nature that emphasizes design thinking, engineering, and entrepreneurship.

The middle school program is still under development. There are no formal CS courses in 6th to 8th grade. Instead, smaller coding projects, such as animating infographics or presentations, are integrated into other core subjects.

Lower school technology courses begin in second grade and continue through fifth grade (fifth grade is technically considered part of middle school). From their website: "Lower school technology courses focus on introducing and developing emerging computing skills, including keyboarding, digital storytelling, conceptual programming, search and online research strategies, media literacy, ethical online behavior, and digital communication and collaboration skills" [22]. The classes meet once a week for an hour in the Media Lab. According to their computer science director, the lower school is currently more of a technology course than a coding class, but moving forward, teachers will place greater emphasis on CS.

- **2nd:** Keyboarding, basic word processing, learning about devices, and navigating the web. Multimedia projects and Scratch are creative outlets.
- **3rd:** Keyboarding, email, Internet safety and ethics, research, organization, word processing, and the creation of communications for audiences other than oneself. Projects include Quicktime movies, Google Earth quests, and class blogs. Programming skills are reinforced with interactive Scratch projects.
- **4th:** Keyboarding, using technology to communicate and collaborate, presentations, basic networking, saving to central servers, and file management. Projects include Quicktime movies, animations, and blog posts.
- **5th:** Internet research, organization, email/Internet safety and ethics, advanced word processing, spreadsheet creation and graphing, Scratch programming (iteration, conditionals, and variables).

### 2.1.2 Public High Schools

Examining the top fifteen STEM-ranked public high schools, as ranked by the U.S. News in 2015 [23], provides insight into the state of computer science in some of the nation's top science and engineering 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 [17].

Table 2.5: Thomas Jefferson High School Graduation Requirements [24]

| **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** |

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.

#### 2.1.2.1. 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" [24]. 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 2.5. Computer science courses, including the graduation requirement, fall under "electives." It is important to note that TJ's graduation requirements are largely similar to Newman's.

#### 2.1.2.2. 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 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" [25].

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

#### 2.1.2.3. 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 [26].

#### 2.1.2.4. 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 [23].

The high school operates on a rotating block schedule; all classes meet Monday for one period, and classes meet on either Tuesday/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 2.6.

#### 2.1.2.5. 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 magnet strands in business, computing sciences and engineering, fine arts, health science and technology, or media. Students take strand-specific courses during the "magnet" elective block (highlighted in orange in Table 2.7). Students who choose the "Computing Sciences and Engineering" select between

TABLE 2.6: Edison Academy: Electronics and Computer Engineering Technology (ECET) Program of Study [27]

| | |
|---|---|
| 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 2.7: Four Year Plan for Westside HS Computing Sciences and Engineering [28]

| **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 |

one of three programs: Computer Programming, Computer Maintenance, and Engineering. The courses comprising these programs are outlined in Table 2.8. The school reportedly uses the four year engineering curriculum developed by Project Lead the Way [28].

TABLE 2.8: Westside HS Computing Sciences and Engineering Magnet Courses [28]

| **Program** | **9th** | **10th** | **11th** | **12th** |
|---|---|---|---|---|
| Computer Programming | Principles of Information Technology/ Principles of Art, A/V Technology and Communication | Pre-AP Computer Science 1 (Python) or Computer Programming | 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) |
| Computer Maintenance | 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 |
| Engineering Design | Introduction to engineering Design | Principles of Engineering | Digital Electronics | Engineering Drafting and Design |
| Engineering Practice | 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.

#### 2.1.3.1. MIT

Massachusetts Institute for Technology (MIT) 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 [29].

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 [29].

#### 2.1.3.2. 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 [30].

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" [30]. These subjects are closely aligned with the CSTA benchmarks for upper level computer science.

### 2.1.4 Code Schools

Around the country code schools are emerging in response to the growing need for software engineers, a need that is not being met by colleges and universities. Code schools are akin to traditional trade schools; employment is the primary focus, and career placement drives the curriculum. For that reason, code schools offer a different approach to computer science than traditional education. Some of the differences may include: strong emphasis on project-based learning, web development languages (JavaScript, Ruby, etc.), greater emphasis on applied rather than theoretical CS, group collaboration using Git and version control, portfolio creation, and interview preparedness.

Hack Reactor is perhaps one of the most well-known and rigorous code schools. Their website states that "students build apps from the ground up and start businesses before they are even out of the program" [31]. Their website showcases projects like drones, Android fitness role playing games, web data visualizations, and more. According to Ian deBoisblanc, a Hack Reactor student in San Francisco, the program is mostly project-oriented and has three overarching themes: learning how to learn, learning by doing, and learning how to work together (Git workflow). The course covers foundational CS topics: algorithms, Big-O notation, functional programming, recursion, data structures, a little bit of networks and computer architecture, classes, polymorphism, APIs, and lots of HTML and CSS. Students also spend an extended period of time on a project they take over from another student, emphasizing the need to write code that is organized and human-readable. Finally, they work on a mini ‘thesis’ as the capstone of the experience.

Operation Spark, detailed in the next section [32], is a local example of a code school. Like Hack Reactor, they use JavaScript, and explore web design, functional and object-oriented programming, and portfolio development.

On the CS educational spectrum, code schools could be considered at the applied, ‘concrete’ end, while theoretical, academic computer science taught or employed in universities might represent the other end of the spectrum. K-12 education needs to find its place in this continuum by identifying and articulating its own academic mission. Is the objective to prepare students for college, to graduate students who are ready for coding employment, or to merely give students a flavor of the diverse applications of CS? The answer is likely a blend of the above, and straddling the pedagogical approaches is the best way to prepare students for a variety of future experiences in computer science.

### 2.1.5 Local Initiatives

#### 2.1.5.1. Independent Schools

Metairie Park Country Day School (MPCDS) is currently in a similar boat as Newman. MPCDS currently offers an introductory and AP CS course in the upper school. Teachers in the middle school are exploring ways to integrate Scratch and Arduinos into the curriculum. Several teachers have also taken an Arduino course as a means to explore the possibility of physical computing.

According to the Tech Director at McGehee School, Margaret Ann Minihan, McGehee currently offers some computer science in grades 3-12. Flor Serna, the founder of Electric Girls, taught coding, making, wearable tech, and electronics to middle school students using tools like Arduinos and Makey Makeys. Beginning next year, however, Flor will be teaching at St. Martin’s, another independent school in the region with a standout making and coding curriculum.

#### 2.1.5.2. Public/ Charter Schools

Benjamin Franklin High School currently offers CS as an elective. This year there is one section of AP Computer Science and two sections of Introduction to Computer Science that has mostly 9th graders. According to the CS teacher, the Intro course is a collection of exercises from the internet or created in-house. Topics include (limited) programming, two-wheeled robot programming, Raspberry Pi coding, a paper on a controversial topic related to CS, basic Excel skills, Photoshop/Illustrator/InDesign, and some beginning cryptography and security.

Lusher Charter High School is currently offering engineering and computer science electives using the Project Lead the Way curriculum. "Computer Science Principles" serves as the introductory computer science course and is aligned with the AP CS Principles. The CS and engineering courses include [33]:

- Introduction to Engineering (1 year, 1 unit): This course offers "hands-on opportunities for students to develop technical solutions to meet real world needs using 3-dimensional computer aided design software to develop models and create hardware based on their models."
- Principles of Engineering (1 year, 1 unit): "This survey course uses 3D modeling software and VEX robotics equipment...Topics include mechanisms, energy, statics, materials, and kinematics. Pre-req: Introduction to Engineering"
- Computer Science and Software Engineering (1 year, 1 unit): "Students create apps for mobile devices, automate tasks in a variety of languages, and find patterns in data. Students collaborate to create and present solutions that can improve people's lives, and weigh the ethical and societal issues of how computing and connectivity are changing the world. This course aligns with the AP Computer Science Principles course."
- Engineering Level IV - Design and Development (1 year, 1 unit): "Engineering Design and Development (EDD) is the capstone course in the PLTW high school engineering program. It is an engineering research course in which students work in teams to design and develop an original solution to a valid open-ended technical problem by applying the engineering design process."

#### 2.1.5.3. Alternative Programs

NOLA Code is an after school K-8 program targeting public/charter schools. The program is free to students on free/reduced lunch. NOLA Code introduces programming using Scratch and Code.org exercises.

Operation Spark is a non-profit that hopes to combat generational poverty by offering software engineering training to low-income, out of school, and unemployed youth in New Orleans. They also offer a limited number of seats for paying applicants to participate in their coding bootcamp. Their website outlines the skills covered in this program [32]:

Fundamentals of Computing, HTML, CSS and JS

- Understanding Hardware, OS, applications, data, and the file system
- Terminal Basics: pwd, ls, cd, mkdir, touch

- Markdown: interpreting markup languages
- HTML: Structure, Text, Lists, Tables, Images, Links, Forms
- CSS Styling: Colors, Text, Boxes
- CSS Positioning: responsive/mobile, grids, frameworks
- Git Fundamentals: clone, add, commit, push
- JS Fundamentals: values, operators, flow control, loops, arrays, objects,functions

Functional JavaScript

- Functional JS (underscore): reimplement each, map, first, last, shuffle, contains, filter/select, every, any
- JavaScript in the browser environment (DOM interface, localStorage, window, events, Ajax) & jQuery
- Advanced HTML: video, audio, canvas, data attributes
- Simple design patterns & micro-utilities: module pattern (for creating private vars), a testing library, a dependency injection system, eventing library, templating library

Object Oriented JavaScript

- OO JS: just decorator pattern, functional instantiation, Object.create
- Using Chrome Developer Tools

Job Skills & Building a Portfolio Site

- Advanced Implementations
- Using Google Maps, Stripe, Soundcloud & other 3rd party API's
- Brief introduction to using Backend-as-a-Service (Parse, Firebase)

## 2.2 National Standards

### 2.2.1 Computer Science Teachers Association

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 a 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 [34]. 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 F.

### 2.2.2 International Society for Technology in Education

The International Society for Technology in Education (ISTE) has compiled a set of standards for computer science educators [35]. These standards are listed in Appendix G 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 Computing at School

Computing at School (CAS) is a community of computer science educators and industry professionals with over 150 regional hubs in the United Kingdom. This community is passionate about computer science education, and in 2012, CAS published a set of standards titled, "Computer Science: a Curriculum for Schools" [36], which are listed in Appendix H.

CAS breaks the standards into five subject areas: algorithms, programs, data, computers, and communication &internet. These benchmarks are broken into "Key Stages":

- Key Stage 1 (K-2nd)
- Key Stage 2 (2nd-5th)
- Key Stage 3 (5th-9th)
- Key Stage 4 (9th-12th)
- Post K-12 education

### 2.2.4 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.

| **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, cybersecurity, 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 2.9: Comparison of AP Computer Science Exams [37]

#### 2.2.4.1. 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. 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, 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" [38].

This shortcoming is particularly stifling 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 [39], 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" [40]. 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 course was the need to "increase the number of students interested in and prepared for success in computer science and other STEM fields" [41]. The course does not specify a specific programming language, but instead encourages students to develop "personally relevant artifacts" [42], whether these are web apps, digital music, animations, or mobile games. "The course is unique in its focus on fostering students to be **creative**" [42] (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" [41]. 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.

## 2.3 Online Curriculum

A non-exhaustive list of online coding tools and curriculum can be found in Appendix D. 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 [43].

Many of the elementary-level tools (Scratch, Tynker, SNAP!) can be used in more sophisticated 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.

### 2.3.1 Massive Open Online Courses

There is great diversity in the number of online courses available to upper school students thanks to the rise of massive open online courses MOOCs. A multitude of computer science courses and subjects can be found on sites such as Coursera, Udemy, Kahn Academy, Udacity, edX, and others. Only a few, however, are optimized for high school instruction and include student tracking, rigorous assessments, standards and benchmarks, or content structured for a typical upper school schedule. The relevant platforms are explored below.

#### 2.3.1.1. Kahn Academy

Kahn Academy offers two "computing" tracks: computer programming and computer science. The former uses Processing.js and JavaScript to cover the following topics (in order): basic statement syntax, variables, math operators, assignment operators, strings, functions, logic and conditionals, loops, arrays, objects, and object-oriented JS. Student tracking is possible through the teacher backend. The courses include 40 talk-throughs, 35 challenges, and 10 projects, which reportedly comes out to about 15 hours of curriculum [44]. The modules include:

- Intro to JS: Drawing & Animation
- Intro to HTML/CSS: Making webpages
- Intro to SQL: Querying and managing data
- Advanced JS: Games & Visualizations
- Advanced JS: Natural Simulations
- HTML/JS: Making webpages interactive
- HTML/JS: Making webpages interactive with jQuery
- Meet the professional

Kahn Academy's "computer science" track includes modules in algorithms, cryptography, information theory, and the internet, all of which require an understanding of CS coding fundamentals. Taken with the computer programming track, Kahn Academy could offer a possible source for self-guided instruction at Newman.

#### 2.3.1.2. Codeacademy

Codeacademy is very similar to Kahn Academy in assessment and content, and students are immediately creating web visualizations or content in the browser. One difference is Codeacademy's click-through instructional style, which may feel more cumbersome than the video instruction offered by Kahn Academy. Units take approximately 8 hours to complete and include:

1. Web Fundamentals - HTML/CSS (20 lessons)
2. Python (36 lessons)
3. JavaScript (28 lessons)
4. jQuery (6 lessons)
5. PHP (11 lessons)
6. Make a Website (5 lessons)

7. Make an Interactive Website (9 lessons)
8. Ruby (30 lessons)

While Codeacademy and Kahn Academy offer student tracking and a diversity of topics, online schools such as CodeHS, Edhesive, and Global Online Academy simulate the experiences of rigorous, in-class instruction aligned to nationally-recognized benchmarks. **Global Online Academy** 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

For a small number of students, Global Online Academy is a cost effective—$500 per student per semester—solution to offering limited CS exposure. This platform, however, is not scalable as a solution to meet a possible CS graduation requirement. Beyond a single section, the courses are no longer cost effective, in addition to the fact that registering more than 2-4 students is likely impossible due to competition from other schools.

#### 2.3.1.3. Edhesive

Edheisve is an online platform optimized to help upper school teachers and administrators get CS programs off the ground. From their promotional materials, "Our program is an all-in-one blended online solution that enables a school to provide Intro and AP Computer Science to its students, even without an AP CS teacher on staff. It includes College Board-approved curriculum with video lessons, automatically-graded assessments, and an active, online forum for content support." Edhesive relies on in-class "coaches" who primarily serve an administrative role. Students get support 7 days/week in an online forum monitored by the online course instructor and TAs. The program appears to be effective. Edhesive boasts of a class average a 3.3 on the 2014 AP CS Exam, while the national average was a 2.95.

#### 2.3.1.4. CodeHS

CodeHS is perhaps the closest competitor to Edhesive. CodeHS offers three courses: Intro to CS, AP CS Principles, and AP CS A. Like Kahn Academy and Codeacademy, CodeHS uses in-browser code-able animations and games (Karel and JavaScript); however, CodeHS is designed specifically for classroom teachers and includes rigorous projects and assessments that are meant to be taken over a school year. For $600/student, it is possible to rely on CodeHS for a fully web based course where teachers oversee progress but don't have an active instructional role. But CodeHS strongly recommends professional development as a means to ensure a trained professional is spearheading the program. Teachers can rely on the web modules in-class, or there are tools for teachers to develop their own projects. For a typical use-case, CodeHS costs $2500-$3000 per section.

When asked about the advantage of CodeHS over Edhesive, a CodeHS sales representative replied,

> "CodeHS curricula definitely provides a lot more opportunity for creativity and flexibility than Edhesive. Their fully online curriculum is released week by week making the course very structured and difficult to change to fit your needs. We've worked with several schools that tried the AP CS course with Edhesive, found the strict pacing to be a problem and ended up using CodeHS."

#### 2.3.1.5. CS50

CS50 is Harvard University's introductory CS course offered through edX, a premier MOOC site used by colleges and universities. The course description states:

> "CS50 - Introduction to the intellectual enterprises of computer science and the art of programming. This course teaches students how to think algorithmically and solve problems efficiently. Topics include abstraction, algorithms, data structures, encapsulation, resource management, security, software engineering, and web development. Languages include C, PHP, and JavaScript plus SQL, CSS, and HTML. Problem sets inspired by real-world domains of biology, cryptography, finance, forensics, and gaming. Designed for majors and non-majors alike, with or without prior programming experience" [45].

The online course is self-paced and requires a substantial time investment. There are 9 problem sets (10 to 20 hours each), and 1 final project.

CS50 AP is a version of this introductory course adapted for the AP CS Principles curriculum framework and offered to high schools for free through the web. Instructional videos, problem sets, and forums are available online to help teachers deliver instruction.

CS50 is noteworthy because it represents the gold standard of online academic CS courses. Harvard is one of the top universities in the world; this curriculum is very rigorous, the instruction is top-notch, and assignments are well-conceived.

This type of academically-demanding course is not for everyone. Like Global Online Academy, CS50 is a possible outlet for highly-motivated, experienced students looking for an advanced, rigorous CS course. It is not, however, a substitute for face-to-face instruction. Like Kahn Academy, CodeHS, or any other MOOC, CS50 should be considered a teaching resource in blended classrooms—possibly for a curriculum outline, for example problem sets, or for videos to mix into lesson plans. The pace and difficulty of the course will turn away many novice but creative students. And like any online program, there is little room for open-ended exploration if strictly following the pre-defined curriculum.

### 2.3.2 Online Pros and Cons

#### 2.3.2.1. Lower and Middle School

In lower school computer science should be experienced through interactive games and artistic platforms that facilitate creativity, something the internet programs do well. Block-based, visual coding platforms like Scratch or Tynker offer immense opportunity for creative expression while delivering foundational computational thinking mindsets. Many of these programs also offer student tracking and online assessments. For both new CS teachers and experienced coders, online coding platforms are an excellent resource in the lower school.

The alternative to online programs at this level are "unplugged" activities—exercises that teach computational thinking without using a computer. Code.org offers several of these activities in their online courses. In "Happy Maps" students act out instructions for and build "algorithms" to help players reach a certain location, and in "Binary Bracelets" students use binary representations of letters in bracelets in order to understand that data can be represented and stored in multiple ways. CSunplugged.org is a website dedicated to providing computational thinking, computer-free exercises. There are a variety of lessons on data compression, searching, and sorting algorithms; binary, image representation, information theory, and more [46].

One of the advantages of "unplugged exercises" according to the authors of, "Computer Science Unplugged..Offline Activities and Games for All Ages," is that "presenting the ideas and issues of computer science [is] often easier to explain with paper and crayons, ordinary materials, and simple activities" [47]. A research paper by the University of Canterbury lists a variety of other advantages: unplugged activities tend to focus on CS concepts rather than programming, students are pulled away from computer screens, exercises are generally cooperative rather than individualistic, and activities are kinaesthetic [48].

The other alternative to online platforms is programmable hardware (hardware tools are listed in Appendix E). Makey Makeys, LEGO robotics, and Bee-Bots are just a few examples. Like the previously mentioned unplugged activities, physical computing can be collaborative, active; these hands-on, open-ended, and creative platforms introduce a variety of other engineering or science concepts. They are another tool with which to diversify CS education and actively engage a broader group of students.

#### 2.3.2.2. Upper School

Online instruction at the upper school offers a variety of advantages when teaching computer science, a rapidly-evolving and highly technical field. At this level the nature of online teaching platforms shifts from games and creative tools to web-based, semester or year-long courses. Content from reputable MOOCs like Kahn Academy and Udacity are up-to-date, taught by subject matter experts, cover a multitude of subjects and programming languages, and frequently are accompanied by interactive games and assessments. The diversity of courses may allow self-directed students to chart academic paths that align with their personal interests.

Online programs are particularly noteworthy in light of the dearth of computer science teachers. Given that the tech industry is struggling to hire programmers for high-paying jobs, it's no wonder that CS teachers are hard to come by [49]. Although there are ample opportunities for professional development, either through the web or in-person workshops, online sites like Edhesive or CodeHS offers schools the ability to move forward without needing CS experts. These platforms provide instruction, assessments, teacher dashboards, and professional development.

The cost to bring a CS program online is another important consideration. For a small number of students, possibly those interested in taking advanced CS courses, Global Online Academy is cost effective: $500 per student per semester. This platform is not scalable, however, as a solution to meet a possible CS graduation requirement. In the case of a required credit, spending $2,500 per section on a CodeHS and recruiting a teacher to "coach" a large number of students is significantly cheaper than hiring a full-time CS teacher.

Another possible alternative to a mandated credit, assuming cost is a significant consideration, is to point students to a variety of free online platforms like Kahn Academy or Coursera, platforms without rigorous built-in assessments but with the ability to track student progress. An end-of-year assessment or senior-year CS project could be used to demonstrate basic mastery of computational mindsets before graduating.

There are a variety of potential concerns related to relying strictly on online courses for CS instruction. The first pertains to student recruitment and retainment—particularly of women, students of color, or other any other student who may not believe CS is relevant to her or his future career paths. Can an online course capture and inspire a broad audience? Finding and retaining students is especially difficult for entirely self-directed courses like those on The Global Online Academy. This platform has attracted a total of five CS students in the last two years. A member of the Tech Club states, "it takes a very specific type of person to do Global Online Academy." Students must be self-motivated and proactive about these courses, something that does not come easily to many students.

For independent schools, the question of value added may be particularly relevant. The technology director at Greenhill School posed the question, "Why would parents spend $30,000 on an education their child could get at home, for free?" In an age where technology and CS play an increasingly important role in the careers of graduating students, this question is particularly salient. Independent schools must keep pace with evolving tools in order to remain relevant and to continue to offer unique, valuable educational experiences.

The value added by teachers cannot be underestimated; high-quality teachers is a critical component of what sets Newman apart. Teachers are also a critically-important component of one of the school's core values: "relationships and rigor." The ability to connect with students, to inspire students, to support and encourage students, is (for now) uniquely human. Even real teachers connected through the web are not comparable to in-person instruction; one student in Newman's Tech Club said, "[you're] not going to Skype with your teacher" (he was referring to the Global Online Academy). It is therefore in Newman's best interest to cultivate good CS teachers rather than shift instruction fully to computers.

The other serious concern with online curriculum is the lack of flexibility. Computer science is one of the most innovative and creative tools of our time. A course that facilitates open-ended exploration, student-driven design questions, and problem sets that match student interest is not possible with a cookie cutter online platform. And if the objective of CS, as some may argue, is to integrate the subject into other disciplines such as English or math, relying on a CS platform that exists solely on the web will prevent cross-discipline interaction. With a course designed in-house, the teacher can play to the specific needs and interests of various students.

Physical computing is another area of interest that is not possible through online platforms. Coding LEDs, robotics, or other electronic systems is possible through tools like Arduinos and Raspberry Pis. There are new ways to interact with the physical world that make coding even more accessible, creative, and approachable. There is currently no way for online programs to offer these types of hands-on, physical programming experiences.

In conclusion, online platforms at the middle and upper school level offer innumerable benefits such as diverse and fast-evolving content taught by subject matter experts, much of which is free and available on the internet. The ability to chart self-directed, variegated learning paths is an incredible resource for self-motivated students. For programs trying to get off the ground, out-of-the-box CS platforms like Edhesive or CodeHS deliver programs and content immediately. These services can also help schools that lack or cannot find talented CS teachers.

But for schools like Newman, schools that set themselves apart with top-notch teaching talent, there is little value added by relying on free, widely-available curriculum. Teachers offer flexibility, tailor exercises to student needs, and enable hands-on, physical computing activities. Perhaps most importantly, teachers communicate enthusiasm for the subject and can build relationships, a core value in Newman's community.

## 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 E for a table of tools.

## 2.5 CS Integration

One of the central questions facing all divisions that are developing computer science programs is: should CS be treated as a stand-alone course, or should it be integrated into existing subjects? Computer science exists in a nebulous space, straddling math, science, engineering, and art. Determining where computer science fits into the curriculum is one of the first steps in developing a longitudinal, sustainable CS plan.

### 2.5.1 Pros and Cons of CS Integration

The decision regarding CS's place in the curriculum must be informed by an examination of the benefits and drawbacks associated with integrating CS into existing subjects. There are a variety of benefits of integration. To begin with, integration eliminates the need to amend the schedule to fit an additional subject. In the upper school some of the scheduling difficulties are related to state-mandated requirements. In 23 states, including Louisiana, computer science cannot be used to meet math or science graduation requirements [7]. Students interested in qualifying for Louisiana's Taylor Opportunity Program for Students (TOPS) scholarship cannot use CS as a substitute for math or science, reducing the time and space for computer science [50]. Incorporating CS into the existing curriculum avoids these obstacles.

In addition, by introducing CS in core subjects like math or science students are guaranteed CS exposure since they don't need to opt-in to a CS elective. Mandated exposure is especially important for women and underrepresented minorities who may have pre-existing ideas about what it means to be a programmer. While a CS graduation requirement may serve the same purpose, adding additional graduation requirements may be an arduous, tedious, or otherwise impossible process. In short, integration alleviates scheduling difficulties while still ensuring all students have exposure to CS.

Integration also forces computer science to be viewed through a cross-disciplinary lens, enabling students to see the broad applicability of coding. Beaver Country Day School in Brookline, Massachusetts, has incorporated coding into every 6-12 course with surprising results. Rob McDonald, the math department head, said, "We're seeing creative ideas from students that teachers hadn't anticipated, and those ideas are leading to projects in areas ranging from engineering to entrepreneurship. Teachers are taking risks, collaborating with one another and identifying new opportunities for interdisciplinary work" [51]. Programming their own graphing calculators in math, animating scenes from Macbeth using code in English, and using code as pencils in art classes are just a few examples of the ways CS infiltrates their curriculum.

Incorporating coding across disciplines also has the potential to build more robust, sustainable CS programs. By spreading the responsibility of coding to multiple teachers in a division, the program can continue to exist and evolve in the event that the tech coordinator or primary CS teacher leaves the school. Particularly in a climate where CS teachers are hard to come by (and potentially hold on to, considering the high paying salaries for programmers), diversifying the levels and modes of instruction could be advantageous.

One of the greatest obstacles to an integrated approach is the need for professional development. Very few teachers, both nationally and at Newman, have an understanding of what computer science entails and why coding should be taught in schools. Fear of, or indifference towards, the unknown makes it difficult to garner teacher buy-in, and without significant support, coding becomes a top-down, burdensome check box that teachers may choose to ignore. An integrated approach breaks down when teachers are not supported, trained, and onboard with coding as a new literacy.

Another major concern raised by teachers is the amount of time coding might absorb from already time-restricted courses. Inevitably there will be overhead associated with teaching the CS skills necessary to complete an inter-disciplinary project. The concern about spending additional time is particularly acute in the upper school where teachers struggle to cram in enough AP content to ensure students score well on national exams. In addition, since math and science curricula are scaffolded across grades and divisions, some teachers worry that reducing subject-specific content will make it difficult to meet the requirements of subsequent courses.

When interviewed about incorporating CS into math, Greg Malis, an upper school math teacher at Newman, said, "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 whether or not he would be interested in integrating CS into his math classes, however, Malis was unsure. He worries about the existing limitations on time that would only be exacerbated by incorporating a new tool into the course.

One advantage that a stand-alone course offers over an integrated approach is that CS instructors have thorough training in their subject area. In upper school the level of instruction necessary to meet CS benchmarks and adequately expose students to coding may be too complex to handoff to teachers in non-CS subjects. At the lower and middle school level, coding platforms (Scratch, Code.org, etc.) are not overly complex so as to prohibit teachers from quickly mastering and integrating coding into curriculum. Nevertheless, CS is a rapidly-evolving, diverse field. Students in all divisions may benefit from taking CS with a subject matter expert whose sole job is to keep up with trends and tools.

If computer science is to be treated as a "core" subject like math and science, stand-alone courses may be the only way to truly deep dive into CS and explore rigorous content. English is integrated into all disciplines, and yet it still receives its own time block in all divisions. In addition, meeting established CS benchmarks may be difficult if coding is diffused across the curriculum. A scaffolded series of stand-alone CS courses may be the best way to ensure students are rapidly progressing, meeting standards, and exploring interesting material.

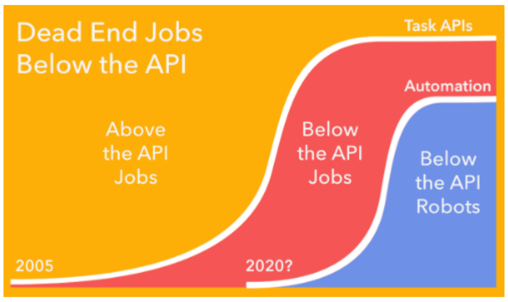
Like all tough problems, there is likely no singular solution to building a robust, sustainable, and rigorous CS program. When possible, integration of CS into existing curriculum helps students see coding as an essential tool in a myriad of fields. Integration also averts scheduling difficulties and may prevent programs from crumbling with the loss of key CS teachers. At the same time, integration is difficult or impossible without adequate professional development and in-school support. An integration specialist or coding support position may alleviate these pain points, but to truly develop a rigorous, top-notch CS program, stand-alone CS courses led by subject matter experts may be the best path forward.

### 2.5.2 CS & Innovation

There is a strong case to be made for integration of computer science into a curriculum emphasizing innovation, engineering, and design. In an age of rapid technological, environmental, socioeconomic, and geopolitical change, the ability to ideate and develop new products or ideas is paramount.

In a post titled, “Replacing Middle Management with APIs,” Peter Reinhardt explores the idea that in the coming decades software will automate millions of jobs, threatening to replace humans with robots. Services like self-driving cars and drone delivery services are already underdevelopment by companies like Uber, Google, and Amazon [52]. “Above the API” (the application programming interface) has come to refer to the people who create and develop instructions for computers as opposed to those who follow or implement them. According to Reinhardt, “as the software layer gets thicker, the gap between Below the API jobs and Above the API jobs widens” [52]. As automation increasingly permeates our culture and workforce, the need to innovate and develop new products, technologies, and services is crucial.

Figure 2.1: Above the API [52]



As our world becomes more complex, the design thinking movement in education and in industry hopes to democratize the innovation process so that more people are endowed with the skills to solve hard problems. Design thinking was developed by IDEO founder David Kelley; it’s essentially a set of mindsets and processes to develop human-centered, innovative solutions [53]. Developing empathy for the user is a core component of the process, as is rapid prototyping and feedback-driven iteration.

The engineering design process is very similar to the mindsets of design thinking: define the problem, brainstorm solutions, test, and iterate. One of the differences is that engineering does not always emphasize a human-centered design approach. As an example, designing a robotic system to land on an asteroid involves constant prototyping, testing, and iteration, but the human perspective may not be as relevant. Nevertheless, design thinking is highly useful to many engineering projects, and engineering is an important discipline in innovation and problem-solving.

In the age of rapid change, the ability to innovate may be a more pressing educational objective than instilling foundational computational thinking CS concepts. Fortunately, CS is one of the most powerful platforms for innovation, and so meeting both learning objectives simultaneously is both possible and potentially advantageous. CS affords the ability to rapidly prototype, perform virtual testing, and continually iterate—pillars of the design thinking process. CS can be used to program hardware (wearable technology, robotics, self-driving cars, 3D printers, etc.) and software solutions (mobile devices, web applications, music, financial data, math, etc.). In short, the possibilities are endless, and CS is an incredible tool to solve some of the world's most difficult challenges.

The obvious examples of software companies that have dramatically changed our daily lives include companies like Uber, AirBnB, Google, Facebook, and Amazon. But there are innumerable examples of innovation through code with altruistic and humanitarian origins. Ushahidi, an open source crowd-sourcing platform to report crisis information during earthquakes or humanitarian disasters, is one example. The project was originally developed to map reports of violence in Kenya after the post-election violence in 2008 [54] and has also been used by the Louisiana Bucket Brigade to map the human and environmental impacts of oil spills [55]. Myo armband, an incredibly versatile wireless wearable technology that tracks arm gestures and motion, may help deaf people communicate by turning sign language into text [56].

Massachusetts Institute of Technology's (MIT) Media Lab is one of the best examples of an integrated, "antidisciplinary" approach to innovation that employs design, engineering, and computer science as tools to develop game-changing technology in myriad fields. Fab labs—spaces with woodworking equipment, laser cutters, 3D printers, electronics, and other shop tools—abound throughout the graduate school, giving students ample access to tools necessary to prototype and develop creative projects. From the MIT website:

> "Actively promoting a unique, antidisciplinary culture, the MIT Media Lab goes beyond known boundaries and disciplines, encouraging the most unconventional mixing and matching of seemingly disparate research areas. It creates disruptive technologies that happen at the edges, pioneering such areas as wearable computing, tangible interfaces, and affective computing. Today, faculty members, research staff, and students at the Lab work in 24 research groups on more than 350 projects that range from digital approaches for treating neurological disorders, to a stackable, electric car for sustainable cities, to advanced imaging technologies that can "see around a corner." The Lab is committed to looking beyond the obvious to ask the questions not yet asked–questions whose answers could radically improve the way people live, learn, express themselves, work, and play" [57].

A tour of the building in April, 2016 revealed that every research group was using CS in some way. The Center for Lifelong Kindergarten is a multidisciplinary group developing solutions to engage people in creative learning experiences. This group is responsible for creative coding platforms like Scratch, App Inventor, LEGO Mindstorms, and the Makey Makey. In the Center for Bits and Atoms (CBA), an "initiative exploring the boundary between computer science and physical science" that "studies how to turn data into things, and things into data" [58], a graduate student is currently developing a 3D world for simulating nano-scale electronic structures using JavaScript and Three.js. In the Synthetic Neurobiology group, researchers are attempting to repair brain disorders "via novel tools for mapping and fixing brain computations" [59]. There are many more examples of the creative application of code used in Media Lab to break the boundaries of what is currently feasible and continually innovate.

The Maker Movement has the potential to wrap all of these disciplines—CS, engineering, design thinking—into a single pedagogical approach that teaches and encourages innovation through hands-on, project-based learning. The Maker Movement was spawned by the emergence of new, easy-to-learn tools like 3D printers, laser cutters, and programmable electronics, tools that are democratizing innovation. Arduinos and Raspberry Pis, simple programmable computers, are making complex electronics projects accessible to tinkerers of all types. Artists, technophiles, programmers, machinists, and DIY-enthusiasts are converging in "makerspaces" to create, share, and invent, while online platforms like Instructables.com allow makers to virtually share step-by-step tutorials for drones, electronic music instruments, 3D printed bobble heads, and more.

Education is increasingly buying in to the Maker Movement. Newman, Patrick F. Taylor, McGehee School, and Lusher High School are just a few examples of schools in New Orleans that are including making in the curriculum. Many schools have already recognized the power of hands-on, experiential learning, an approach backed by research. A study from Purdue showed that students scored higher in science classes where the goal was to design and build a device than students in traditional classrooms [60]. A study from the University of Chicago had similar findings; brain scans revealed that students' sensory and motor-related brain regions were activated when they thought about topics like angular momentum and torque [61]. Making allows students to chart their own paths, to choose their own hard problems, and to actively engage in the exploration of electronics, programming, and art.

By refocusing the educational learning objective from the need to instill computational thinking mindsets to the need to foster creative and innovative thinking skills, integrating CS into design thinking, engineering, and the Maker Movement makes sense. Computer science is an incredibly versatile tool, and using this tool to solve human-centered or open-ended design challenges may be the most effective, valuable, and necessary approach to CS education.

# Chapter 3

# Newman Past and Present

## 3.1 Previous Programs

There hasn't always been a dearth of K-12 CS courses at Newman. In the 1990s computer courses and faculty positions existed in all divisions. From around 1991-1996 Minnette Patton, currently the Assistant Head of Lower School, taught the lower school Computer course. Students in K-5 came to the lab 2-3 times per week, and depending on grade level, students started with Logo or MicroWorlds. Hyperstudio, keyboarding, word processing, PowerPoint, and connecting to internet were also covered in the class. In the upper school, all students had to take a computer applications course to graduate, as mandated by the state of Louisiana. Computer literacy, specifically keyboarding, was a large part of the curriculum. Around 1994 a million dollar technology grant was used to put computers in classrooms. Technology coordinators were hired in each division to help teachers integrate technology into the classroom, while computer teachers continued teaching coding.

By the early 2000s, however, coding had ceased in all divisions. One of the failures of this program according to Dale Smith, the Head of School and a former CS teacher at Newman, was that CS was not integrated into other subject areas, thereby failing to remain relevant.

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 resurgence 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

Susie Toso currently teaches coding during Media, a class that meets once per week for 50 minutes in grades 1st-4th. Media covers the following subjects and tools:

**1st** Students work with coding iPad apps such as Kodable, Scratch Jr., and Tynker. The emphasis is on play and exploration. Kodable topics include: sequencing, loops, conditionals, functions, ints, and Strings.

**2nd** Second grade works with Code.org and has completed Course 1 (rated for ages ages 4-6) and 2 (for ages 6+). From the Code.org website, subjects from Course 1 include sequences, loops and events, collaboration, problem-solving, and internet safety. Course 2 covers conditionals, algorithms, binary code, debugging, and the societal impacts of computing.

**3rd** In third grade students code with Scratch, practice setting up Google Docs, visit websites through Symbaloo, and practice logging in.

**4th** Fourth grade is split between Kodable (sequencing, loops, conditionals, functions, ints, and Strings) and Code.org.

In lower school science Jennifer Williams and Elaine Sevin offer a collaborative 2nd/5th grade LEGO WeDo robotics unit that meets once per month for 50 minutes. The exercise, to control a soccer player and kick a ball into a LEGO goal, simulates real life phenomenon 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. Programmable hardware is also used in 2nd grade science. During the Mars unit, students program Bee-Bots, small robots for teaching sequencing and problem solving.

Students may also get exposure to robotics after school. Once a week students in 3rd-5th grades program LEGO Mindstorms robots to complete challenges. The team also competes in the regional FIRST LEGO League robotics competition.

Computer literacy also takes place in the classrooms and during library time. Typing Pal is currently the purview of the homeroom; teachers assign 10 minute typing exercises twice a week. During library, students learn about conducting internet research, navigating the web, and internet safety. At this time, there is no standardized curriculum or widely-adopted set of computer literacy benchmarks but the LS Newman curriculum is on track to meet CSTA coding standards. Specific program recommendations are discussed in the next section.

#### 3.2.1.1. 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:

**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 C. 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 implementing 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 sacrificed to make room for this new subject?

### 3.2.2 Middle School

There are no CS courses currently offered in the Middle School. Computer science exposure is limited to Coding Club, a small group of students that meets once per week on Wednesdays for approximately 30 minutes. This semester there was not enough student interest to keep the club going, but in previous semesters Kate Loesher, a sixth grade math teacher and the club advisor, reports that students worked through Kahn Academy exercises. Student progress depended highly on the student and whether or not they were interested in coding at home.

A small group of students has the opportunity to learn coding after school. The middle school robotics team programs LEGO Mindstorms EV3 robots using a block-based language akin to Scratch. During Makerspace, which also meets once per week after school, students may get exposure to Scratch and Arduino. Overall, however, these programs are reaching a negligible portion of the student body.

### 3.2.3 Upper School

As of writing, CS exposure at Newman is largely self-guided and limited to a small group of students. A handful of students are enrolled in Global Online Academy, an online platform offering courses in a variety of subject areas (refer to Section 2.3). In the last two years, seven students have enrolled in Computer Programming I and II, and two students have taken Game Theory.

Students in Tech Club meet on a weekly basis in the Makerspace to build and program their autonomous drone. Tech Club evolved from Coding Club, a group formed in 2014 that focused on HTML, Java, and Google Code Jam. According to founding members, Coding Club ostensibly engaged students in extracurricular CS, although like many student groups, participation quickly waned as students got bored with online exercises. There were seven members last year, but the club founder reports that only a couple students actively engaged.

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.

# Chapter 4

# Recommendations

A table of K-12 CS benchmarks tailored specifically for Newman can be found in Appendix A. These benchmarks were compiled using frameworks and curriculum from the CSTA (Appendix F), ISTE (Appendix G), Computing at School (Appendix H), College Board (Section 2.2.4), universities, code schools, and other online resources.

It felt necessary to tailor benchmarks specifically for Newman given some of the shortcomings of existing standards. CSTA benchmarks are narrowly specific in some subject areas, overly vague in others, or excessively pedantic (e.g. "use technology to conduct age-appropriate research"). ISTE and the College Board's AP are restrictive in their sole focus on upper school. By picking from and refining established benchmarks, the list enumerated in Appendix A hopes to provide a concise but thorough set of standards that can be used as the bedrock framework for an aligned, K-12 CS program at Newman.

## 4.1 Lower School

It is exciting to report that Media and LS science currently address the coding benchmarks outlined in Appendix A. The following recommendations explore potential ways improve upon the current program.

Using Media time to explore physical computing may help reach a group of students that responds well to tactile instruction. Robotics also serves to introduce engineering at an early age, in addition to reinforcing CS concepts learned through online coding platforms. Hardware programming occurs in 2nd and 5th grade science through Bee-Bots or LEGO WeDo, but time with these technologies is limited and could be augmented by Media. Unfortunately, Susie Toso reports that the Media block is barely enough time to get students logged in and coding; adding hardware only complicates setup time. The need for funding or additional professional development may be another barrier to physical computing.

Kodable is currently one of the main online platforms used to teach coding in the LS, and so a critical evaluation of its strengths and weaknesses should inform its use in the classroom. One of the issues with Kodable is that many of the lessons follow the same format; there is little variability within units (“Loopy Lagoon,” “Condition Canyon,” etc.). While repetition helps to reinforce concepts, the formulaic exercises may develop high proficiency with the game without developing a foundational understanding of the underlying skill.

Another potential weakness of the platform is that it does not visually reinforce terms; the absense of text labels (“functions,” “loops,”, etc.) may be an intentional design decision to accommodate new readers, but developing a coding vernacular is an important step towards identifying themes and ideas across CS. On the plus side, the accompanying lesson plans are very thorough and outline solid learning objectives that teachers can use to reinforce concepts (e.g. “functions help recycle code so there are no repeats”). That said, these teacher resources are dense and could be confusing, particularly for teachers who may be new to CS. Distilling these lesson plans down in a way that is digestible for the intended audience, K-5 students, may not be straight-forward.

The section Asteroidia is perhaps the most problematic section, which calls into question the validity of the platform as a whole. As an experienced programmer, I struggled to determine how the exercise (shooting asteroids) was linked to a computer science concept; after reading the teacher resource, I can say that the link between shooting asteroids and variable assignment is tenuous as best. In general, I found the section to be little more than an overly complex, confusing shooting game with unclear numbers and letters, all of which were supposed to communicate variables, Strings, and Ints.

The important takeaway from this section, a point that is relevant to any lower school coding platform, is that the underlying concepts and learning objectives should match the age-appropriate benchmarks enumerated in Appendix A. Kodable tries to cover middle and upper school subjects in its K-5 platform, with mixed results. Variables (the core learning objective of the aforementioned Asteroidia section) are not a recommended benchmark until middle school because students have not explored this relatively complex subject in math classes. The same goes for Strings, Ints, arrays, and classes (subjects explored in Kodable lessons). Early exposure is certainly not detrimental, and in some cases may be advantageous, but in the case of Kodable, attempting to present advanced material resulted in a confusing asteroid game. Sticking to K-5 benchmarks is likely a better use of coding time, as additional emphasis on loops, sequences, conditional statements, and functions will lay a foundation for future advanced CS coursework.

All of these criticisms aside, Kodable is still a good resources for PK-2 coding. The sections on sequencing, problem decomposition, loops, conditionals, and functions clearly communicate ideas through fun, engaging exercises. By providing additional context to help students identify recurring ideas and themes, which Susie is already doing, and exposing students to additional coding platforms (currently the case), Kodable can be considered one of several tools in the early education coding toolbox.

The other heavily-used platform, Code.org, currently offers a series of impressive online courses. Unlike Kodable, lessons are variegated, and there is opportunity for free play (as opposed to exercises with a single solution). The online curriculum also includes "unplugged" activities, activities that don't require a computer, which is another way to actively engage students in a hands-on experience of CS. In addition, the ability to "see code" may allow curious students to get an early taste of text-based coding, the step beyond block-based programming. Videos from diverse professionals in the field help to communicate at an early age who codes (Women! People of color!) and what coding entails.

Another issue that may be worth exploring is the gap in the LS CS sequence since 5th graders currently do not take Media. Since Susie Toso currently works part-time, she would not be able to cover 5th grade even if there were space in the schedule. The other gap (PK and K) is not as concerning. While coding exercises and platforms exist at this age, there are a limited number of benchmarks to be covered at the lower school level, and beginning before 1st grade may not be a valuable use of time.

As illustrated in the LS faculty brainstorm (Appendix C), there may be *computer literacy* benchmarks, in addition to coding benchmarks, that should be articulated and addressed. While these topics should not necessarily be the purview of Media, establishing a list of computer literacy benchmarks to be distributed between technology, the library, and the classroom may help to ensure that students don't fall through the cracks, as well as help establish clear expectations between teachers. With the hiring of a new LS librarian, this is an opportune time to develop library curriculum that meets some of these computer literacy benchmarks. Conducting internet research, safely using the internet, and creating word processing documents could be just a few examples of important computer-related topics that do not need to take up coding time. Finally, keyboarding is another important computer literacy skill that is covered by classroom teachers. Examining the efficacy of this program (and debating the overall necessity of mandated keyboarding) should be explored by LS teachers and administrators.

One final recommendation is to consider a capstone coding project or assessment prior to 6th grade. This type of project may help to ensure all students are on the same page upon entering middle school. Many online platforms offer assessment tools, but an open-ended Scratch project, for example, could be a more creative way to demonstrate mastery.

Oskie Creech, the coordinator of Design Thinking and Innovation at Newman, suggested the possibility of a final coding project in 5th grade. He currently sees all 5th graders for design thinking and said that one semester could be devoted to coding. Since coding lends itself nicely to open-ended design problems and since there is currently a CS gap in 5th grade, his class may be an optimal place in which to implementing a capstone coding project.

### 4.1.1 Summary

To summarize, Susie Toso has laid a framework for coding in lower school Media, and Jennifer Williams and Elaine Sevin supplement CS content with LEGO WeDo and Bee-Bots in 2nd and 5th grade science. Thanks to the current curriculum, students are leaving lower school with an understanding of the benchmarks enumerated in A.

Moving forward, additional programmable hardware and unplugged activities (refer to Section 2.3 for additional details), if feasible, would help to diversify the instruction. Sticking to Code.org and Scratch (as opposed to Kodable) beyond 2nd grade is recommended. Continuing to ensure that lessons provide verbal reinforcement of concepts, provide context, and draw parallels between platforms is also important (e.g. “a function recycles code,” “loops repeat code—in what other programs did we see loops?”). Developing a set of computer literacy benchmarks that are distributed between technology, library, and the classroom will help to create accountability and set expectations for non-coding computer skills. And finally, exploring the idea of a capstone coding project during 5th Design Thinking would help to ensure all students are ready for CS at the middle school level.

## 4.2 Middle School

Finding a way to meet middle school CS benchmarks outlined in Appendix A is arguably more difficult than in other divisions, thanks to limited space for electives. In this case, integrating CS into existing courses might the easiest way to expose all students to coding. Rather than advocating for a particular implementation strategy, however, this document will explore the pros and cons of a few possibilities.

### 4.2.1 CS Integration

For a more thorough analysis of the pros and cons of CS integration, please refer to Section 2.5.

There are innumerable possibilities for the types of projects that can be incorporated into any discipline. An easy and common integration approach is to have students develop interactive multimedia presentations (complete with sounds and moving characters) using Scratch. Example projects include animating historical events, illustrating book reports, or teaching geology lessons. Scratch gives students a creative, hands-on platform for exploring any subject with code.

Code.org offers middle school curriculum that uses coding as a tool to teach algebra and science. The science curriculum explores life, physical, and earth sciences. Modules include modeling and simulating ground water, ecosystems, and planetary systems using code. In the algebra lessons students design a video game and simultaneously use coding to explore order of operations, the Cartesian plane, functions, and word problems [62]. The non-profit highlights one advantage of integration: "By shifting classwork from abstract pencil-and-paper problems to a series of relevant programming problems, Code.org **CS in Algebra demonstrates how algebra applies in the real world**, using an exciting, hands-on approach to create something cool" [62].

If Newman decides to take the route of integration, a series of scaffolded projects distributed between grades and subjects could help meet the benchmarks enumerated in Appendix A. Two projects each year (in math, science, English, art, or history) would likely be sufficient to meet these standards. An integration coordinator would likely be needed to build relationships with interested teachers, develop projects inline with existing curriculum, provide professional development, and assist in the execution of the coding projects. While there would initially be substantial work on the front-end, this approach has the potential to be more effective and robust in the long-run.

Another possibility discussed this year by Peter Cline, the Head of Middle School; Lisa Swenson, the 6th grade science teacher and class dean; and Jenna deBoisblanc was the possibility of flipping Lisa's "general" science course into a course that strongly incorporates coding into the curriculum. "Exploratory Design," a coding, engineering, and design thinking elective currently offered at Greenhill School in Dallas, Texas, may be a good a model the evolution of this course. From the "Exploratory Design" description:

> "This hands-on exploratory class takes a thematic approach to the presentation and solution of problems. Students are formally introduced to problem-solving processes from the fields of computer science, engineering, and science as they seek to develop products and models that address a specific set of problems. Students work with a set of guiding processes drawing from Design Thinking, Engineering and Design, Computational Thinking and Next Generation Science Processes."

Topics include "Mission to Mars," "The Deep Abyss," "Feeding the World," and "Energy for All"—topics that explore remote sampling, remote communication, contamination, planetary geology, pressure, modeling, and renewable energy. Coding, engineering design, and design thinking are emphasized throughout the curriculum.

Arduinos, easily-programmable computers for creating electronics projects, could be the perfect tool to complement existing science curriculum; currently Lisa covers electricity, magnetism, geology, water, weather, and climate. Arduinos are designed to program electronics and can used to introduce electricity, simple circuits, breadboards, LEDs, voltage, current, and resistance. In addition to electricity, Arduinos are great tools for environmental science labs. "Environmental

Monitoring with Arduino" is a free, online PDF that contains projects such as "Humidity, Temperature & Dew Point Display" and "Measuring Water Conductivity." Public Lab, a non-profit based in New Orleans and New York that develops DIY environmental sensors for communities and education, relies heavily on Arduino. Some of their "Open Water" sensors include Riffle Water Monitoring (open hardware device that measures some of the most common water quality parameters), and the The Homebrew Sensing Project (uses spectroscopy to indicate water contamination) [63]. These are just a few examples of projects that could be adapted to water, weather, or climate curriculum.

Given the versatility of Arduinos—their ability to sense the environment, control motors, blink LEDs, generate sounds, etc.—they are a perfect prototyping tool for design thinking projects. Lisa's class currently participates in a design thinking exercise called, "Light It Up," in which students develop design thinking solutions using LEDs and simple circuits. With a few lines of code, students could program Arduinos to blink, to turn on with the push of a button, to fade in response to a sliding switch, to act like a timer, to control rainbow LEDs, and so much more. A platform like Arduino could take prototypes in this design thinking project to the next level while exposing students to CS.

A science course with heavy integration of coding has the advantage of reaching every student in the 6th grade and giving students substantial CS exposure. This course could easily meet the middle school benchmarks in Appendix A while still covering environmental content, electronics concepts, and design thinking projects.

Another advantage of a coding course with Arduinos is that Arduinos could serve as a perfect bridge from LS to MS, and MS to US computer science. In addition to programming Arduinos in a text-based coding language (based on C++), students may also program Arduinos in Scratch, a block-based coding platform currently used in the lower school. The ability of this platform to be used in both programming formats creates a seamless coding transition between divisions.

The issue of PK-12 science alignment is one major concern brought up in response to this proposed approach. Jennifer Williams, the Department Chair Lower School Science, articulated this concern:

> "The Lower School science portion of the curriculum adjusted our scope and sequence three years ago to pick up electricity and magnetism, so Lisa could have a more environmental, earth focus for sixth grade science. She is still teaching these concepts, since there would be a gap of four years of students not having any electricity or magnetism studies with the movement of these topics into LS. This movement of these concepts was difficult for us to absorb, since students in grades Pre-K through 4th do not have science every day of a school week. It also meant that our study of the moon had to be cut in 2nd grade to make the change for electricity. Susan also had to cut and/or shorten

> aspects of her program, too, for the incorporation of magnetism. While we do not mind addressing these concepts in LS, please realize that it does have effects on our tight spiral in the LS."

Another concern with a coding-heavy 6th grade science course is that science already struggles to cover a vast amount of content, and adding coding would only exacerbate restrictions on time. Even if the ultimate objective is to preserve existing science curriculum but augment exercises and projects with coding, there may be overhead associated with teaching CS, overhead that may come at the expense of existing science content. As one MS science teacher put it, "I have often contended that there is far more science to teach in middle school than there is time to teach science. Why limit the science that the science department attempts to convey?"

This concern about coverage emanates from a more fundamental disagreement: the issue of whether computer science belongs in science at all. "I'm not exactly sure why the science department is the proper venue for introducing either electronics or coding into the curriculum... We actually once had a computer science department and if there seems to be a renewed interest in that arena, I'm not sure why science is the place to integrate." While this position is not universally-held among faculty (Greg Malis: "The ultimate goal of science is to formulate hypotheses, test, and debug. That, in a nutshell, is what coding's about"), it is clear that tensions exist regarding the possibility of inserting CS into science courses (or for that matter, any existing course).

Considering that this concern has the potential to derail any attempts at integration, I'd like to offer a personal opinion in response to the argument that CS doesn't belong in science. I should note that the use of the first person pronoun in this paragraph is a deliberate choice that diverges from previous attempts to retain objectivity; the choice is meant to emphasize that my view is both debatable and potentially contentious.

That said, of the scientists I currently know in grad school or in industry—a current physicist grad student and former classmate, my environmental science college friend who is building geology maps, my brother doing a psychology PhD at NYU, my best friend getting a PhD at MIT Media Lab, and a fellow physics major who is currently working for an engineering firm that develops drones—every single one of these scientists writes code on a regular basis either to automate testing in Matlab, analyze data with Python, or code embedded systems with C++. In my mind, the argument that programming doesn't belong in science is akin to the argument that calculators don't belong in math, or that computers don't belong in science. Coding is a tool just like Excel is a tool, just like computers and calculators are tools, all of which are used every day in science to conduct research and actively explore the world. Coding has the added benefit of being able to model complex systems virtually, to simulate the impacts of climate change or chemical reactions, to model and predict disease transmission, to illustrate and interactively manipulate the laws of physics. Coding not only has the potential to aid and accelerate the progress of science, it has the potential to improve, through models and hands-on coding projects, the experience of science education. And finally, the computational thinking mindsets associated with CS have wide-ranging benefits, including the development of better, more logical problem solvers. For these reasons, I would argue that coding is a highly-relevant, necessary practice in the modern scientific method and in science education.

Now, should the onus of teaching CS fall solely on the back of science? No. Nor should science blindly adopt coding without reviewing the impacts on K-12 science curriculum. Just as computers (or reading and writing for that matter) are universal tools employed and refined in all subjects, so too should coding be the responsibility of all subjects. Integration must occur with the active participation of teachers and with buy-in from department chairs to ensure cross-division content alignment is preserved. Ultimately, successful integration should augment existing content and projects in a manner that is both seamless and unobtrusive. But regardless, before CS integration can take place, it is important to understand and embrace the benefits of coding to each department and division.

### 4.2.2 Art Elective

Of the nine INDEX schools interviewed for this report, the most common way that computer science was implemented in the middle school was through the arts. Six of eight schools with middle school CS programs offered either a required or elective coding course in the arts rotation. At Greenhill School all 5th and 6th grade students take "Exploratory Design," a design thinking and engineering CS course that lasts one trimester and shares an elective block with another art credit. At St. Margaret's in San Juan Capistrano, CA, 6th and 7th graders elective block is comprised of 1/4 art, 1/4 music, 1/4 health & wellness and 1/4 CS. Winchester Thurston in Pittsburgh, PA, has a (required) one-trimester elective each year of middle school, in addition to a maker elective.

There are innumerable possibilities for an engaging art-themed coding course. Using Arduinos to create wearable tech LED clothing, electronic music instruments, or interactive robots are just a few examples. As discussed in the previous section, Arduinos could serve as a perfect bridge from LS to MS, and MS to US computer science because they can be programmed in a text-based coding language (based on C++) as well as with Scratch, a block-based coding platform. But regardless of the platform, coding is a very creative tool that could easily supplement the current arts rotation.

The most obvious advantage of offering CS through the arts rotation at Newman (a trimester, rotating sequence) is the relatively low barrier to entry; teacher buy-in is not as critical as it might be through an integrated approach. A standalone course negates the need for professional development and can be implemented almost immediately.

Perhaps the greatest argument against implementing CS through the arts is that students who take band, a year-long art elective, would miss out on exposure to CS. In addition, enrollment in existing performing and visual arts courses would be impacted, particularly if (as this document recommends) CS is required every year of middle school. As discussed in Section 2.5, a standalone course is dependent on finding a talented CS teacher.

### 4.2.3 Club/ Advisory/ Flex

Club time is a 30-minute, once-weekly opportunity for students to experience a non-traditional or non-academic subject. Current clubs include cooking, chess, chemistry, film, ping-pong and more. Since all middle school students take club, allocating this time block to coding has the advantage of offering CS exposure to all students in 6th-8th grades. Teachers or advisors could use online coding platforms like Code.org, platforms that require minimal teacher input in order to deliver CS curriculum. Online platforms would reduce the need for substantial, division-wide professional development to deliver instruction.

When asked about the possibility of using club time for middle school-wide CS instruction, Kate Loescher, the sixth grade science teacher who previously ran the Middle School Coding Club, seemed optimistic about the idea but worried about the need for professional development. During club Kate assigned Kahn Academy exercises to students, but did not actively monitor their progress. If coding were mandatory, however, she is confident that active engagement from teachers could lead to meaningful student progress. In order to prepare for questions, Kate spent a little time working through Kahn Academy exercises. She also directed advanced students to help students with questions. It was not difficult to facilitate the club and help students through problems, but Kate pointed out that she likely has more CS exposure than the rest of the faculty.

One of the difficulties of club is the disjointed timing. In Maker Club, as an example, it's difficult to maintain continuity between projects given that the club meets once per week. The thirty minute time block also makes it difficult to accomplish substantial projects; reorienting and eating snack takes half of the block. The other concern is that club time may be an integral part of the Newman middle school experience. The relative merits of clubs must be considered in order to make a decision about its role the CS implementation solution.

Advisory time is another potential avenue for CS. Advisory meets three times a week, shares the same time block as club, and runs into similar issues: short blocks and the potential to displace an important middle school experience.

Another possibility is to do coding during flex time. All grades in 6th-8th have flex during the same block as PE but on alternating days. Flex has the advantage of meeting more regularly (every other school day), of meeting for 45 minutes instead of 30, and for splitting up each grade into a different time block. The advantage of a separate time blocks is that it might enable one CS subject-matter expert to attend multiple classes throughout the day in order to support teachers and students.

The problem with using flex time for CS, however, is that this time block is currently used by students to get extra help or work on homework. Repurposing this time may not be to the advantage of students, particularly those who are falling behind. If, however, the only requirement is that students finish a particular online coding unit (a self-paced but teacher-tracked unit) before the end of the trimester, students may be able to spend time coding when other assignments are light and vice versa. In addition, meeting the Newman benchmarks in Appendix A may only take an equivalent of a trimester worth of flex time, a potentially manageable burden when spread throughout the year.

In any of these solutions, substantial analysis and input from teachers, students, and administrators will be necessary before moving forward. Professional development will be necessary to ensure that teachers feel comfortable with CS instruction since they would be required to teach, or at the very least, monitor student progress. Developing or executing highly-engaging CS exercises may also be more difficult since the teachers implementing the projects will not be subject-matter experts. There are a variety of online programs, however, that offer fun, engaging coding experiences, in addition to providing student tracking and online assessments (refer to Appendix D).

### 4.2.4 A Path Forward...

Taking a middle school CS program from zero to sixty is neither advisable nor possible; instead, small, easy-to-prototype projects that gradually build teacher support and interest are the appropriate path forward. To that end, working with early adopters to integrate a few easy, unobtrusive coding projects into existing curriculum at each grade level should be the focus next year. Lisa Swenson has already expressed interest in implementing coding in her classroom, and as discussed in previous sections, design thinking projects like "Light It Up" may be the perfect avenue for coding with Arduinos. Scratch multimedia presentations that can replace PowerPoint are another example of an easy introduction to CS integration. With a few successful projects under the belt, discussions may evolve about how to offer CS in a more substantial way that meets the benchmarks in Appendix A, either through a standalone course or through integration. In any scenario, however, a knowledgeable coding advocate will be necessary to keep the ball rolling, to educate teachers about CS, and to help develop coding projects.

| **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

## 4.3 Upper School

Unlike the middle school schedule, the upper uchool sequence affords time and space for computer science electives. The rotating eight block schedule means there are 32 possible credits, although Newman students rarely graduate with greater than 28. The student handbook stipulates that students must complete 24 academic credits to graduate (refer to Table 4.1).

The elective space (two required elective graduation credits) is one possible avenue for CS courses, as is the Art Department. Offering CS courses with titles like, “Creative Coding” or “Virtual Reality Animation” may allow courses to fit into the arts rotation. Offering coding courses as arts electives may attract a broader audience, particularly girls, than traditional CS electives. The perception of arts courses as less rigorous than science electives (for better or for worse) may also encourage greater participation. On the other hand, offering coding through the Art Department may impact the enrollment of other art courses, particularly if a variety of CS electives are implemented. Since many students choose to take art courses to fill the two credit additional elective requirement, however, the impact on existing art classes may be the same regardless of whether CS is offered as a general elective or an art elective. Nevertheless, finding the appropriate home for computer science—in art, science, or math—should be discussed by teachers and administrators.

### 4.3.1 1st Tier

As a first step, implementing an introductory CS elective with broad appeal is paramount. There are many different possible introductory courses in a variety of different programming languages, but the priority of this course must be to hook students. Highly-engaging, fun, hands-on, and topical projects should communicate foundational concepts outlined in Appendix A. Students should leave the class understanding the broad applicability of CS, as well as the potential for innovation and creativity in a multitude of fields.

One possible introductory course, The Art of Making, is outlined in Appendix B. This hands-on course strongly emphasizes coding, but also teaching fundamentals in 3D printing, laser cutting, and electronics. The course relies on Processing, a visual programming language designed for artists, and Arduino, a programmable platform for interactive electronics projects. By leveraging these tools together, the possibilities are only limited by the imagination: LED dresses that sparkle when an accelerometer registers movement, drones, electronic music instruments, interactive data visualizations, and so much more.

### 4.3.2 2nd Tier

There are already students at Newman who are ready for a second tier CS course. A course titled "Creative Coding" (outlined in Appendix B) could offer the same creative freedom as the introductory course but with advanced, second tier topics (enumerated in Appendix A). In addition to the previous benchmarks from tier one, tier two introduces object-oriented programming structures (either classes or prototypes depending on the language).

While this document strongly recommends against offering AP Computer Science A, at least in the early stages of CS, Creative Coding lends itself nicely to the new AP CS Principles framework (for a full discussion of the benefits and drawbacks of teaching each AP CS course, please refer to Section 2.2.4). AP Computer Science Principles (the new course) may offer both the national recognition of an AP while still preserving the creative and innovative component of CS. This AP is young, so time and experimentation may be necessary to determine the course's true value.

One requirement of offering the AP is that the course be year-long in order to adequately prepare students for the exam. The benchmarks outlined in Appendix A could be covered in a single semester. An AP CS Principles course should cover the benchmarks listed in tiers two and three. If a year-long course at the second tier is not possible or advisable, the AP could be offered as a third tier course.

### 4.3.3 3rd/4th Tier

A third tier of CS electives will ensure that motivated students can continue to develop as computer scientists. There are infinite possibilities for courses that can meet the advanced benchmarks in Appendix A. Future courses will likely depend on the instructor, student preferences, and total enrollment. Like Creative Coding, any of the courses listed below could likely be adapted to the AP CS Principles framework. These course titles could also be used as a fourth tier by ensuring content and projects meet the benchmarks listed in Appendix A. Potential titles include:

- Web Design and Development
- Software Engineering
- Mobile Apps
- Robotics with Python
- Virtual Reality with JavaScript
- Advanced Physical Computing

Absent from this list are courses titled, “Advanced Data Structures,” “AP CS A,” and “Object Oriented Programming Using C++”—courses currently taught at other high schools that fail to relate CS to other disciplines, that emphasize CS theory without making room for creative expression, and that, at least historically, have failed to reach a broad, diverse audience. Creative, open-ended, project-driven courses that teach equivalent computational thinking concepts should comprise 3rd and 4th tier CS.

### 4.3.4 Innovation Ecosystem

Section 2.5 makes an argument for the incorporation of CS into curriculum that focuses on design, engineering, making, and innovation. The tools and technology of Newman’s Makerspace are the perfect venue for exploring innovation through code. This track system begins with The Art of Making (refer to Appendix B for a sample syllabus), which serves as the introduction to coding, design thinking, 3D printing, laser cutting, and electronics—the foundation of all courses to evolve out of the Makerspace.

At the next level students may choose between two tracks: computer science or engineering design. Each track may comprise of scaffolded tiers of courses. Although the tracks emphasize one particular discipline over the other, both tracks continue to expose students to coding, design thinking, and engineering design.

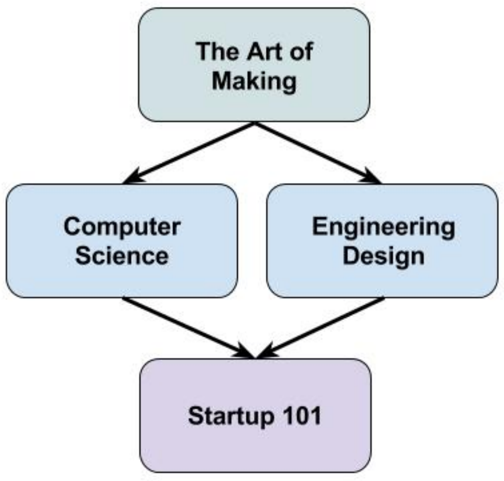
Potential courses in each track are outlined below.

Computer Science

1. Creative Coding
2. Web Design and Development
3. Software Engineering

Engineering Design

FIGURE 4.1: Innovation Ecosystem



1. 3D Art and Design / Tech Theater / Environmental Engineering Design
2. Industrial Design
3. Robotic Systems

Finally, the tracks reconverge with Startup 101, a course focusing on the ideation, creation, testing, and marketing of unique products and services. The course builds on skills acquired in computer science, engineering, 3D modeling, and design, but adds the element of entrepreneurship. Students will write business models and develop marketing strategies. By the end of the course, students will have an understanding of pathways to innovation, in addition to an understanding of how successful products are launched.

### 4.3.5 Required Exposure

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. The importance of CS and the dire need for coding experience is explored in Chapter 1. Of the nine INDEX schools interviewed about CS, the three schools with the most impressive CS programs all had graduation requirements of approximately one semester (refer to Section 2.1.1).

That said, there are a few potential problems associated with a graduation requirement. Mandated credits in the Upper School are currently impacting scheduling for certain students; additional requirements would place even greater restrictions on the courses students can take. In addition, a graduation requirement would necessitate the hiring of additional faculty to teach CS courses, which may prove to be difficult. Staffing difficulties, however, should not prevent the enactment of a graduation requirement. The ubiquity of professional development opportunities and online resources should enable non-CS teachers (possibly math or science) to pick up a language. Hiring and training a new teacher, however, may take time.

If a required graduation credit is not feasible within a reasonable timeline, mandating CS integration in the Upper School should be considered. Re Each department could be required to expose students to CS through a project at some point during 9th-12th grade, or class deans could be responsible for ensuring that at least one course exposes students to CS each year. Ideally these coding experiences would be scaffolded or coordinated in such a way that students still meet the minimum CS requirements outlined in Appendix A. An integration coordinator would be necessary to help non-CS teachers brainstorm and implement project ideas, coordinate between departments and classes to ensure that each class gets necessary exposure to coding, and develop content that meets necessary graduation requirements.

The drawbacks and benefits of CS integration are explored in more depth in Section 2.5. One of the primary drawbacks of integration is placing additional burdens on upper school teachers who are already struggling to meeting curriculum requirements. The lack of understanding of CS and the need for professional development is another primary drawback. A integration coordinator, however, may help to mitigate both of these obstacles and would allow all students to get CS exposure without requiring additional graduation requirements.

### 4.3.6 Online Education

For a full analysis of online platforms, their merits and shortcomings, please refer to 2.3.

There are several examples where online programs add clear value to Newman's CS offerings. Allowing platforms like Global Online Academy or CodeHS to fill niche needs—such as allowing self-directed students to take specific upper level courses—is cost-effective and grants students more options. In addition, using Kahn Academy or other online resources in blended classrooms as a crutch or supplement may bolster in-class instruction.

This document, however, does not recommend the strict reliance on online CS platforms, especially for meeting a potential graduation requirement. By the very nature of being "cookie cutter" and purely virtual, online platforms cannot offer the same "rigor and relationships" as curriculum spearheaded by a teacher. Hooking students, inspiring students, communicating the importance and broad applicability of CS, and ensuring that students recognize the creative and innovative potential of coding—these must be the priorities of a CS program at Newman. CS courses will not compete with other disciplines at Newman without the enthusiasm and relationships created by and with teachers, the personally-relevant curated projects, or the face-to-face communication. In short: Newman is known for its top-notch pedagogy; CS, a critically important field in the 21st century, cannot be the exception.

All of that said, hooking students is the priority, after which students may want more freedom to explore advanced topics on their own. Using online CS programs as a second or third tier course may be advisable, but only if resources or hiring difficulties prohibit an in-person CS experience.

### 4.3.7 Next Steps

Next year two new courses will be offered in the Makerspace—"The Art of Making" and "Creative Coding"—which will lay the groundwork for potential upper school courses in engineering and computer science. Offering these courses as art electives to a small group of students will give time to prototype projects and give insight into the best type of CS instruction that meets the Newman CS benchmarks outlined in A. In time, questions related to graduation requirements, additional courses and tiers, offering the AP, and hiring/ staffing needs can be addressed. For now, drumming up student interest and building teacher support is paramount in order to build a sustainable and resilient program.

# Chapter 5

# Conclusion

A national review of the status of computer science K-12 education spawns a variety of questions that must be addressed as a CS program develops at Newman. In the Lower School, those questions relate to the balance of coding vs. computer literacy, classroom vs. Media time, physical vs. virtual instruction, and game-based vs. open-ended creative platforms. In the Middle School, which is less amenable to additional electives, the question is: should CS be integrated into the classroom or offered as a standalone course? How will we make space for a new "core" subject in an already tight schedule? And for the Upper School, the questions relate to curriculum—should Newman offer an academic approach to CS akin to an introductory course at the university level, or should an upper school program place greater emphasis on the creative, innovative, and applied coding process from industry and code schools? What are the merits of offering the AP? And finally, can online CS instruction replace classroom instruction?

While many of these questions can be debated, the dire need for computer science at Newman cannot. Students are graduating without valuable, marketable, and creative computational thinking skills. If Newman is to remain a competitive, top-notch independent school, it must keep pace with this evolving field.

Fortunately, even the independent schools with standout programs—schools like Winchester Thurston and Greenhill—are relatively new to CS. Using nationally established benchmarks (CSTA, ISTE, AP, etc.), learning from the experiences of top tier schools and institutions, and leveraging new tools and technologies, Newman has the ability to craft a holistic K-12 CS plan and establish itself as a local and national leader in computer science education.

# Appendix A

# Appendix: Newman CS Benchmarks

Below is a list of benchmarks for PK-12 computer science at Newman. These benchmarks were compiled using CSTA (Appendix F), ISTE (Appendix G), AP (Section 2.2.4), CAS (Appendix H), and other online resources and benchmarks. They are broken into grade ranges. Benchmarks beyond "US Tier 1" are optional standards that may be met by students who express interest in computer science.

**PK-3rd**

1. Understand that coding is the process of giving precise, step-by-step instructions to a computer in order to achieve particular goals.
2. Use a block-based program to experiment with sequencing and loops.
3. Decompose a problem into smaller problems.
4. Experience coding with programmable hardware (Bee-Bots, LEGO robotics, Makey Makey, etc.).

**3rd-6th**

1. Understand that coding is the process of giving precise, step-by-step instructions to a computer in order to achieve particular goals.
2. Decompose a problem into smaller problems using code.
3. Demonstrate an understanding of loops and their utility in programming.
4. Successfully use conditional statements in a program.
5. Use a block-based program to create a unique coding artifact.

6. Experience coding with at least two platforms (e.g. Scratch and LEGO Mindstorms) and identify similarities between them.

7. Experience coding with programmable hardware (Bee-Bots, LEGO robotics, Makey Makey, etc.) and draw parallels to other coding platforms.

8. Understand how to safely use the internet and safeguard personal information.

9. Understand how to conduct age-appropriate research on the internet using reliable, credible sources.

**6th-9th**

1. Demonstrate an understanding of loops and their utility in programming.

2. Successfully use conditional statements in a program.

3. Decompose a problem into smaller problems using code.

4. Demonstrate an understanding of primitive data types and the utility of variables in coding.

5. Write functions and explain their utility in computer programs.

6. Experience programming with at least one text-based programming language.

7. Effectively use modeling and simulation with a programming language to solve real-world problems.

8. Create a unique coding artifact with block or text-based programming language.

9. Articulate a variety of potential applications of coding in various fields and disciplines.

10. Understand how to safely and ethically use the internet (particularly as it relates to social media and new platforms for online interaction).

11. Describe strategies for determining the reliability of information found on the Internet.

**US Tier 1 (minimum graduation requirement)**

1. Understand abstraction and its importance in problem solving.

2. Demonstrate proficiency with standard data structures: primitive data types, Strings, Arrays (1-D and 2-D).

3. Understand variables and variable scope.

4. Effectively write functions and pass parameters, and understand the importance of functions in abstraction.
5. Effectively use sequential, conditional, and iterative control structures.
6. Demonstrate proficiency with Boolean expressions.
7. Use a standard programming language to create a unique coding artifact.
8. Understand binary representation- representing names, objects or ideas as sequences of 0s and 1s.
9. Evaluate programs written by others for readability and usability.
10. Understand types of programming errors and develop effective strategies for debugging code.
11. Use markup languages (HTML, CSS) and JavaScript to develop simple web pages.
12. Identify a myriad of jobs and disciplines related to CS.
13. Describe ethical issues that relate to computers and networks (e.g., security, privacy, ownership, and information sharing).

**US Tier 2**

1. Understand encapsulation and demonstrate proficiency with object oriented programming (prototypes or classes)
2. Develop an understanding of inheritance and polymorphism.
3. Understand and implement recursion.
4. Understand hexadecimal notation and demonstrate and understanding of data representation at the machine level.
5. Understand command line basics.
6. Use various debugging and testing methods to ensure program correctness.
7. Analyze and manipulate large data sets using a programming language.
8. Use Application Program Interfaces (APIs) and libraries to facilitate programming solutions.
9. Use version control (Git) to collaborate on group projects.
10. Effectively use modeling and simulation to solve real-world problems.

11. Differentiate among open source, freeware, and proprietary software licenses.
12. Understand and explain how the internet functions.
13. Identify existing cybersecurity concerns and potential options to mitigate these issues.

**US Tier 3**

1. Understand more advanced data structures - array lists, linked lists, stacks, queues, etc.
2. Design and implement searching (linear and binary) and sorting algorithms (bubble, selection, insertion, and merge sort)
3. Analyze algorithms by considering complexity, efficiency, aesthetics, and correctness.
4. Explain how sequence, selection, iteration, and recursion are building blocks of algorithms.
5. Classify problems as tractable, intractable, or computationally unsolvable.
6. Explain the value of heuristic algorithms to approximate solutions for intractable problems.
7. Understand data compression: loss-less (Huffman coding) and lossy compression algorithms (example JPEG).
8. Describe a variety of programming languages available to solve problems and develop systems.
9. Understand basic computer architecture: CPU, storage (e.g. hard disk, main memory), input/output (e.g. mouse, keyboard).
10. Understand networks: basic components (e.g., servers, file protection, routing, spoolers and queues, shared resources, and fault-tolerance) and issues that impact network functionality (e.g., latency, bandwidth, firewalls, server capability).

**US Tier 4**

1. Understand and implement advanced data structures: trees, graphs, hash tables, etc.
2. Understand two's complement signed integers.
3. Understand problems of using discrete binary representations (e.g. quantization, representing fractions).
4. Demonstrate knowledge of varied software development models (MVC) and project management strategies.
5. Explore multiple programming paradigms (e.g. functional vs. object-oriented).

6. Deploy principles of security by implementing encryption and authentication strategies.
7. Demonstrate concurrency by separating processes into threads and dividing data into parallel streams.
8. Use persistent data storage (writing files, SQL and non-relational databases, etc.).

# Appendix B

# Appendix: Sample CS Syllabi

## TIER 1: THE ART OF MAKING

This course is a hands-on introduction to the technology and mindsets of making. Projects will introduce students to coding, engineering, and design thinking by leveraging tools like Arduinos, 3D printers, laser cutters, and more. Wearable tech, line-following robots, and electronic music instruments are just a few examples of potential projects created in this course.

### Course Objectives

1. Know how to safely operate the tools of the Makerspace, including the laser cutter, 3D printers, and CNC.
2. Develop basic 3D modeling proficiency with CAD tools.
3. Develop an introductory-level understanding of object oriented programming, including data types, variables, control structures, and algorithms.
4. Communicate coding's enormous potential for innovation and creativity, as well as its applicability to a diverse set of careers and disciplines.
5. Understand and apply design thinking principles in the creation of digital or physical prototypes.

### Grade Breakdown

- 10% - Class Participation
- 10% - Homework
- 30% - Project 1 - Wearable Tech Design Challenge

- 30% - Project 2 - Electronic Music Design Challenge
- 10% - TED Tech Talk
- 10% - Online Web Portfolio

**Overview**

| **Week** | **Overview** | **Description** | **Benchmarks** |
|---|---|---|---|
| 1 | Introduction to Arduino | Variables, primitive data types, binary data representation, functions | US1.1, US1.2, US1.3, US1.4, US1.8 |
| 2 | Sensors and Logic | Sensors, Boolean logic (if/else statements), Serial Monitor | US1.6 |
| 3 | Control Structures | Arrays, control structures, libraries | US1.2, US1.5 |
| 4-7 | Wearable Tech Design Challenge | Students will work in groups to design and code a human-centered, interactive wearable tech solution to address a design challenge. | US1.7, US1.9, US1.10 |
| 8-9 | Introduction to 3D Modeling | Students will work with Autodesk 123D Design to explore the fundamentals of 3D modeling and 3D printing. | |
| 10 | Introduction to Laser Cutting | Vector programs (Inkscape), bitmap tracing, etching and cutting | |
| 11-14 | Electronic Music Instruments Design Challenge | Students will work in teams to develop innovative electronic music instruments using all of the tools explored during the course: 3D printing, laser cutting, Arduinos, electronics, and coding. | US1.7, US1.9 |
| 15-16 | Online Portfolio | Students work with HTML, CSS, and JavaScript to develop simple online portfolios. | US1.7, US1.11 |

**TED Tech Talks**

TED Tech Talks are weekly lectures given by students to educate the class about the diverse applications of coding, important computer science topics not covered in the course, or additional programming languages that are not explored in The Art of Making. TED Talks cover benchmarks US1.12 and US1.13. Potential topics include: internet infrastructure, cybersecurity, internet ethics; coding applications to music, art, medicine, video games, math, science research, or other disciplines; and information about languages like Max/Jitter, Processing, Python, Java, etc.

## TIER 2: CREATIVE CODING

Creative coding is an introduction to computer science that employs computer programming as a medium for creative expression and innovation. Students develop creative solutions to design thinking challenges using languages like Processing, a visual programming language designed for artists, and Arduino, a programmable platform for building electronics projects. Web animations, data visualizations, wearable technology, interactive LED art, and electronic music instruments are just a few potential projects explored in this hands-on course.

### Course Objectives

1. Develop a deeper understanding of object oriented programming, including classes, objects, methods, polymorphism, and inheritance.
2. Successfully collaborate on a programming project using version control (Git).
3. Explore JavaScript libraries and APIs for creative, interactive web development.
4. Manipulate and analyze (through visualization) large data sets.
5. Communicate coding's enormous potential for innovation and creativity.

### Grade Breakdown

- 5% - TED Tech Talks
- 10% - Class Participation
- 10% - Homework
- 20% - Project 1 - Generative Art
- 20% - Project 2 - Data Visualization
- 25% - Project 3 - Virtual Reality
- 10% - Online Portfolios

### Overview

| **Week** | **Overview** | **Description** | **Benchmarks** |
|---|---|---|---|
| 1-2 | Introduction to Processing | Variables, primitive data types, functions, arrays, Boolean logic, control structures, drawing, color, hexadecimal notation | US2.4 |

| | | | |
|---|---|---|---|
| 3-4 | Classes in Processing | Classes, objects, methods, inheritance, polymorphism, abstract classes | US2.1, US2.2 |
| 5-7 | Generative Art | Students will develop an interactive, generative art project that implements a unique class. | |
| 8 | Introduction to Processing.js | HTML, CSS, Processing.js, data analysis | US2.8, US2.3, US2.6 |
| 9-10 | Data Visualization | Students will find a personally-relevant data set and code an interactive data visualization on the web. | US2.7, US2.10, US2.5 |
| 10 | Git | Version control, push, pull, branches, merges, collaboration | US2.5, US2.9 |
| 11 | Three.js and A-Frame | Introduction to libraries for creating JavaScript VR | US2.8 |
| 12-15 | Virtual Reality | Students will work in teams to create a virtual reality world. Teams must collaborate and demonstrate proficiency with version control (Git). | US2.9 |
| 16 | Online Portfolio | Students work with HTML, CSS, and JavaScript to develop simple online portfolios. | |

**TED Tech Talks**

TED Tech Talks are weekly lectures given by students to educate the class about a highly creative, innovative application of computer programming. TED Talks cover benchmarks US2.11, US2.12, and US2.13.

# Appendix C

# Appendix: Lower School Computer 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:

**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?

The number of times particular subjects came up in responses to these questions is recorded in Table C.1.

TABLE C.1: LS Computer Brainstorm Summary

| Times Recurring | Subject |
|---|---|
| 14 | professional development |
| 14 | word processing |
| 12 | internet research |
| 12 | safety and ethics |
| 7 | saving |
| 6 | keyboarding |
| 6 | time restrictions |
| 5 | balance between CS and coding |

| | |
|---|---|
| 5 | email |
| 5 | iPad |
| 5 | troubleshooting |
| 4 | images and video |
| 4 | PowerPoint |
| 3 | appropriate |
| 3 | college prep |
| 3 | creativity |
| 3 | printing |
| 2 | confidence |
| 2 | CS in context |
| 2 | curriculum development |
| 2 | integrate into existing curriculum |
| 1 | network |

# Appendix D

# Appendix: Online Curriculum

The following table is compiled from EdSurge[64] and Code.Org[65]. For full descritions and side-by-side comparisons, visit Code.org's 3rd Educator Party Resources.

Table D.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 E

# Appendix: Hardware Tools

TABLE E.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 F

# Appendix: CSTA Benchmarks

The Computer Science Teachers Association (CSTA) K-12 Computer Science Standards (revised 2011) [34] 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 F.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 F.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 F.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 F.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 F.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 G

# Appendix: ISTE Standards

The International Society for Technology in Education (ISTE) compiled the “Computer Science Standards for Educators” [35]. 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 H

# Appendix: Computing At School Benchmarks

Computing at School (CAS) is a community of computer science educators and industry professionals with over 150 regional hubs in the United Kingdom. This community is passionate about computer science education, and in 2012, CAS published a set of standards titled, "Computer Science: a Curriculum for Schools" [36], which are enumerated below.

CAS breaks the standards into five subject areas: algorithms, programs, data, computers, and communication &internet. These benchmarks are broken into "Key Stages":

- Key Stage 1 (K-2nd)
- Key Stage 2 (2nd-5th)
- Key Stage 3 (5th-9th)
- Key Stage 4 (9th-12th)
- Post K-12 education

**KEY STAGE 1**
ALGORITHMS

1. Algorithms are sets of instructions for achieving goals, made up of pre-defined steps *the 'how to' part of a recipe for a cake+.
2. Algorithms can be represented in simple formats [storyboards and narrative text].
3. They can describe everyday activities and can be followed by humans and by computers.
4. Computers need more precise instructions than humans do.
5. Steps can be repeated and some steps can be made up of smaller steps.

PROGRAMS

1. Computers (understood here to include all devices controlled by a processor, thus including programmable toys, phones, game consoles and PCs) are controlled by sequences of instructions.

2. A computer program is like the narrative part of a story, and the computer's job is to do what the narrator says. Computers have no intelligence, and so follow the narrator's instructions blindly.
3. Particular tasks can be accomplished by creating a program for a computer. Some computers allow their users to create their own programs.
4. Computers typically accept inputs, follow a stored sequence of instructions and produce outputs.
5. Programs can include repeated instructions.

DATA

1. Information can be stored and communicated in a variety of forms e.g. numbers, text, sound, image, video.
2. Computers use binary switches (on/off) to store information.
3. Binary (yes/no) answers can directly provide useful information (e.g. present or absent), and be used for decision.
4. Computers are electronic devices using stored sequences of instructions.
5. Computers typically accept input and produce outputs, with examples of each in the context of PCs.
6. Many devices now contain computers

COMMUNICATION AND THE INTERNET

1. That the World Wide Web contains a very large amount of information.
2. Web browser is a program used to use view pages.
3. Each website has a unique name.
4. Enter a website address to view a specific website and navigate between pages and sites using the hyperlinks.

**KEY STAGE 2**
ALGORITHMS

1. Algorithms can be represented symbolically [flowcharts] or using instructions in a clearly defined language [turtle graphics].
2. Algorithms can include selection (if) and repetition (loops).
3. Algorithms may be decomposed into component parts (procedures), each of which itself contains an algorithm.
4. Algorithms should be stated without ambiguity and care and precision are necessary to avoid errors.
5. Algorithms are developed according to a plan and then tested. Algorithms are corrected if they fail these tests.
6. It can be easier to plan, test and correct parts of an algorithm separately.

PROGRAMS

1. A computer program is a sequence of instructions written to perform a specified task with a computer.
2. The idea of a program as a sequence of statements written in a programming language [Scratch]
3. One or more mechanisms for selecting which statement sequence will be executed, based upon the value of some data item
4. One or more mechanisms for repeating the execution of a sequence of statements, and using the value of some data item to control the number of times the sequence is repeated

5. Programs can model and simulate environments to answer "What if" questions.
6. Programs can be created using visual tools. Programs can work with different types of data. They can use a variety of control structures [ selections and procedures].
7. Programs are unambiguous and that care and precision is necessary to avoid errors.
8. Programs are developed according to a plan and then tested. Programs are corrected if they fail these tests.
9. The behaviour of a program should be planned.
10. A well-written program tells a reader the story of how it works, both in the code and in human-readable comments
11. A web page is an HTML script that constructs the visual appearance. It is also the carrier for other code that can be processed by the browser.
12. Computers can be programmed so they appear to respond 'intelligently' to certain inputs.

DATA

1. Similar information can be represented in multiple. Introduction to binary representation [representing names, objects or ideas as sequences of 0s and 1s].
2. The difference between constants and variables in programs.
3. Difference between data and information.
4. Structured data can be stored in tables with rows and columns. Data in tables can be sorted. Tables can be searched to answer questions. Searches can use one or more columns of the table.
5. Data may contain errors and that this affects the search results and decisions based on the data. Errors may be reduced using verification and validation. Personal information should be accurate, stored securely, used for limited purposes and treated with respect.

COMPUTERS

1. Computers are devices for executing programs.
2. Application software is a computer program designed to perform user tasks.
3. The operating system is a software that manages the relationship between the application software and the hardware
4. Computers consist of a number of hardware components each with a specific role [e.g. CPU, Memory, Hard disk, mouse, monitor].
5. Both the operating system and application software store data (e.g. in memory and a file system)
6. The above applies to devices with embedded computers (e.g. digital cameras), handheld technology (e.g. smart phones) and personal computers.
7. A variety of operating systems and application software is typically available for the same hardware.
8. Users can prevent or fix problems that occur with computers (e.g. connecting hardware, protection against viruses)
9. Social and ethical issues raised by the role of computers in our lives.

COMMUNICATION AND THE INTERNET

1. The Internet is a collection of computers connected together sharing the same way of communicating. The internet is not the web, and the web is not the internet.

2. These connections can be made using a range of technologies (e.g. network cables, telephone lines, wifi, mobile signals, carrier pigeons)
3. The Internet supports multiple services (e.g. the Web, e-mail, VoIP)
4. The relationship between web servers, web browsers, websites and web pages.
5. The format of URLs.
6. The role of search engines in allowing users to find specific web pages and a basic understanding of how results may be ranked.
7. Issues of safety and security from a technical perspective.

**KEY STAGE 3**
ALGORITHMS

1. An algorithm is a sequence of precise steps to solve a given problem.
2. A single problem may be solved by several different algorithms.
3. The choice of an algorithm to solve a problem is driven by what is required of the solution [such as code complexity, speed, amount of memory used, amount of data, the data source and the outputs required].
4. The need for accuracy of both algorithm and data [difficulty of data verification; garbage in, garbage out]

PROGRAMS

1. Programming is a problem-solving activity, and there are typically many different programs that can solve the same problem.
2. Variables and assignment.
3. Programs can work with different types of data [integers, characters, strings].
4. The use of relational operators and logic to control which program statements are executed, and in what order
   - Simple use of AND, OR and NOT
   - How relational operators are affected by negation *e.g. NOT (a¿b) = a $\leq$ b+.
5. Abstraction by using functions and procedures (definition and call), including:
   - Functions and procedures with parameters.
   - Programs with more than one call of a single procedure.
6. Documenting programs to explain how they work.
7. Understanding the difference between errors in program syntax and errors in meaning. Finding and correcting both kinds of errors.

DATA

1. Introduction to binary manipulation.
2. Representations of:
   - Unsigned integers
   - Text. [Key point: each character is represented by a bit pattern. Meaning is by convention only. Examples: Morse code, ASCII.]
   - Sounds [both involving analogue to digital conversion, e.g. WAV, and free of such conversion, e.g. MIDI]
   - Pictures [e.g. bitmap] and video.

3. Many different things may share the same representation, or “the meaning of a bit pattern is in the eye of the beholder” *e.g. the same bits could be interpreted as a BMP file or a spreadsheet file; an 8-bit value could be interpreted as a character or as a number].
4. The things that we perceive in the human world are not the same as what computers manipulate, and translation in both directions is required [e.g. how sound waves are converted into an MP3 file, and vice versa]
5. There are many different ways of representing a single thing in a computer. [For example, a song could be represented as:
   - A scanned image of the musical score, held as pixels - A MIDI file of the notes - A WAV or MP3 file of a performance]
6. Different representations suit different purposes [e.g. searching, editing, size, fidelity].

COMPUTERS

1. Computers are devices for executing programs
2. Computers are general-purpose devices (can be made to do many different things)
3. Not every computer is obviously a computer (most electronic devices contain computational devices)
4. Basic architecture: CPU, storage (e.g. hard disk, main memory), input/output (e.g. mouse, keyboard)
5. Computers are very fast, and getting faster all the time (Moore’s law)
6. Computers can ‘pretend’ to do more than one thing at a time, by switching between different things very quickly

COMMUNICATION AND THE INTERNET

1. A network is a collection of computers working together
2. An end-to-end understanding of what happens when a user requests a web page in a browser, including:
   - Browser and server exchange messages over the network
   - What is in the messages [http request, and HTML]
   - The structure of a web page - HTML, style sheets, hyperlinking to resources
   - What the server does [fetch the file and send it back]
   - What the browser does [interpret the file, fetch others, and display the lot]

3. How data is transported on the Internet - Packets and packet switching
   - Simple protocols: an agreed language for two computers to talk to each other. [Radio protocols “over”, “out”; ack/nack; ethernet protocol: first use of shared medium, with backoff.]

4. How search engines work and how to search effectively. Advanced search queries with Boolean operators.

**KEY STAGE 4**
ALGORITHMS

1. The choice of an algorithm should be influenced by the data structure and data values that need to be manipulated.
2. Familiarity with several key algorithms [sorting and searching].
3. The design of algorithms includes the ability to easily re-author, validate, test and correct the resulting code.

4. Different algorithms may have different performance characteristics for the same task.

PROGRAMS

1. Manipulation of logical expressions, e.g. truth tables and Boolean valued variables.
2. Two-dimensional arrays (and higher).
3. Programming in a low level language.
4. Procedures that call procedures, to multiple levels. [Building one abstraction on top of another.]
5. Programs that read and write persistent data in files.
6. Programs are developed to meet a specification, and are corrected if they do not meet the specification.
7. Documenting programs helps explain how they work.

DATA

1. Hexadecimal
2. Two's complement signed integers
3. String manipulation
4. Data compression; lossless and lossy compression algorithms (example JPEG)
5. Problems of using discrete binary representations: - Quantization: digital representations cannot represent analogue signals with complete accuracy [e.g. a grey-scale picture may have 16, or 256, or more levels of grey, but always a finite number of discrete steps] - Sampling frequency: digital representations cannot represent continuous space or time [e.g. a picture is represented using pixels, more or fewer, but never continuous] - Representing fractional numbers

COMPUTERS

1. Logic gates: AND/OR/NOT. Circuits that add. Flip-flops, registers (**).
2. Von Neumann architecture: CPU, memory, addressing, the fetch-execute cycle and low-level instruction sets. Assembly code. [LittleMan]
3. Compilers and interpreters (what they are; not how to build them).
4. Operating systems (control which programs run, and provide the filing system) and virtual machines.

COMMUNICATION AND THE INTERNET

1. Client/server model.
2. MAC address, IP address, Domain Name service, cookies.
3. Some "real" protocol. *Example: use telnet to interact with an HTTP server.]
4. Routing
5. Deadlock and livelock
6. Redundancy and error correction
7. Encryption and security

# 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] Chris Granger. Coding is not the new literacy. 2015. URL http://www.chris-granger.com/2015/01/26/coding-is-not-the-new-literacy/.

[9] 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.

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

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

[12] 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.

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

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

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

[16] 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/.

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

[18] 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.

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

[20] Nueva School. Nueva philosophy and overview. . URL http://www.nuevaschool.org/academics/philosophy-and-overview.

[21] Nueva School. Scientific foundations (stem). . URL http://www.nuevaschool.org/academics/upper-school/stem.

[22] Nueva School. Information technology. . URL http://www.nuevaschool.org/academics/lower-school/technology#grade-3.

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

[24] 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.

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

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

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

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

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

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

[31] Hack Reactor. Hack reactor curriculum. URL http://www.hackreactor.com/curriculum.

[32] Operation Spark. Operation spark bootcamp curriculum. URL https://operationspark.org/.

[33] Lusher Charter School. Lusher charter school: High school curriculum. 2016. URL http://lusherschool.org/hs-curriculum-course-catalog.html.

[34] 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.

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

[36] Computing at School Working Group. Computer science: a curriculum for schools. March 2012. URL http://www.computingatschool.org.uk/data/uploads/ComputingCurric.pdf.

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

[38] 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/.

[39] 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.

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

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

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

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

[44] Programming curriculum overview. *Kahn Academy*. URL https://www.khanacademy.org/coach-res/reference-for-coaches/teaching-computing/a/programming-curriculum-overview.

[45] Harvard's cs50 introduction to computer science. *edX*. URL https://www.edx.org/course/introduction-computer-science-harvardx-cs50x.

[46] CSunplugged.org. Cs unplugged exercises. URL http://csunplugged.org/text-compression/.

[47] Witten Bell and Fellows. Computer science unplugged. 1998. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.114.7908&rep=rep1&type=pdf.

[48] Tim Bell et al. Computer science unplugged: school students doing real computing without computers. 2009. URL http://www.computingunplugged.org/sites/default/files/papers/Unplugged-JACIT2009submit.pdf.

[49] Donna St. George. High school students are all about computers but get little instruction in computer science. April 2014. URL https://www.washingtonpost.com/local/education/high-school-students-are-all-about-computers-but-get-little-instruction-in-computer-science/2014/04/23/13979eda-c185-11e3-bcec-b71ee10e9bc3_story.html.

[50] Louisiana Office of State Financial Assistance (LOSFA). Taylor opportunity program for students (tops). 2016. URL http://www.osfa.state.la.us/MainSitePDFs/TOPS_OPH_brochure_8-11.pdf.

[51] Robin Burks. This is how an innovative school integrated coding into all of its courses. *Tech Times*, January 2015. URL http://www.techtimes.com/articles/24689/20150105/this-is-how-a-innovative-school-integrated-coding-into-all-of-its-courses.htm.

[52] Peter Reinhardt. Replacing middle management with apis. February 2015. URL http://rein.pk/replacing-middle-management-with-apis.

[53] Kaan Turnali. What is design thinking? *Forbes*, May 2015. URL http://www.forbes.com/sites/sap/2015/05/10/what-is-design-thinking/#553c036b3c18.

[54] Ushahidi. About ushahidi. URL https://www.ushahidi.com/about.

[55] Louisiana Bucket Brigade. iwitness pollution map. URL http://map.labucketbrigade.org/.

[56] Anthony Cuthbertson. Armband controller translate sign language into text. *Newsweek*, February 2016. URL http://www.newsweek.com/myo-armband-controller-translates-sign-language-text-428043.

[57] MIT Media Lab. Quick facts. . URL http://www.media.mit.edu/about.

[58] MIT Media Lab. Center for bits and atoms. . URL http://cba.mit.edu/about/index.html.

[59] MIT Media Lab. Synthetic neurobiology. . URL http://www.media.mit.edu/research/groups/synthetic-neurobiology.

[60] Rick Callahan. Purdue study: Hands-on learning better. *JC Online*, February 2009. URL https://www.purdue.edu/discoverypark/learningcenter/assets/pdfs/HandsOn.pdf.

[61] Jann Ingmire. Learning by doing helps students perform better in science. *UChicagoNews*, April 2015. URL http://news.uchicago.edu/article/2015/04/29/learning-doing-helps-students-perform-better-science.

[62] Code.org. Middle school. . URL https://code.org/educate/curriculum/middle-school.

[63] Public Lab. Nonprofit initiatives. . URL https://publiclab.org/wiki/nonprofit-initiatives#Open+Water.

[64] 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.

[65] Code.org. 3rd party educator resources, . URL https://code.org/educate/3rdparty.