November 2019

Supporting the Algebra I Curriculum with an Introduction to Computational Thinking Course

Michelle M. Laskowski
Louisiana State University and Agricultural and Mechanical College

Follow this and additional works at: https://digitalcommons.lsu.edu/gradschool_theses

Part of the Algebra Commons, and the Educational Technology Commons

Recommended Citation
https://digitalcommons.lsu.edu/gradschool_theses/5037

This Thesis is brought to you for free and open access by the Graduate School at LSU Digital Commons. It has been accepted for inclusion in LSU Master's Theses by an authorized graduate school editor of LSU Digital Commons. For more information, please contact gradetd@lsu.edu.
Supporting the Algebra I Curriculum with an Introduction to Computational Thinking Course

Michelle M. Laskowski
SUPPORTING THE ALGEBRA I CURRICULUM WITH AN 
INTRODUCTION TO COMPUTATIONAL THINKING COURSE

A Thesis
Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Natural Sciences

in

The Interdepartmental Program in Natural Sciences

by
Michelle Marie Laskowski
B.S, Louisiana State University and Agricultural and Mechanical College, 2015
December 2019
Acknowledgments

I would like to thank my parents for providing the absolute best guidance and upbringing that has always motivated me to work hard and persevere through any challenge. Thank you for all your ongoing support, even from afar.

I would also like to say thank you to Professor James Madden and Professor Juana Moreno for serving on my committee and providing me guidance and confidence. I would especially like to thank Fernando Alegre for his mentorship and faith in my abilities to teach the Introduction to Computational Thinking Course, this opportunity has reignited my childhood interests in computers and ultimately in computer science. And thank you to Professor Frank Neubrander for his years and years and years of subtle guidance, wisdom and endless support. I have truly been blessed with my opportunities, encouragement, mentorship and support at Louisiana State University.
# Table of Contents

Acknowledgements .............................................................................................................. ii

Abstract ....................................................................................................................................... iv

Chapter 1. Introduction ........................................................................................................... 1

1.1. Introduction .................................................................................................................. 1

1.2. History of The Computer .............................................................................................. 1

1.3. Literature Review .......................................................................................................... 10

1.4. Advanced Placement’s Computer Science Principles .................................................. 32

1.5. Pearson’s MyMathLab .................................................................................................. 34

1.6. LSU Pathways Project STEM Enrichment .................................................................... 34

1.7. An Overview of the Introduction to Computational Thinking Course ....................... 36

Chapter 2. Supporting Algebra I through the Introduction of Computational Thinking Course .................................................................................................................. 41

2.1. An Overview of the Additional Projections .................................................................. 41

2.2. Embedded Projects ....................................................................................................... 44

2.3. Additional Projects ........................................................................................................ 49

Chapter 3. Conclusions .......................................................................................................... 51

Appendix A. Introduction to Computational Thinking Course Outline ................................. 54

Appendix B. Embedded Projects .......................................................................................... 56

Appendix C. Additional Algebra I Standards Aligned Projects ........................................... 65

References .................................................................................................................................... 79

Vita ................................................................................................................................................ 84
Abstract

The Louisiana Workforce Commission predicts a 33.6% increase in computer science and mathematical occupations by 2022 and the Bureau of Labor Statistics foresees a 16% increase in computer scientists from 2018-2028. Despite these opportunities for job and financial security, the number of Louisiana students enrolled in a nationally accredited computing course is less than 1%, compared to national leaders California and Texas which have 3% and 3.8% of students respectively. Furthermore, the international assessments of mathematical literacy, PISA and TIMMS, both report American students continue to fall further behind their international peers in mathematics achievement.

This thesis rejects these statistics as definitive and attempts to contribute to an expansion of the mathematical libraries of a computational thinking course that a teacher could use to support a standards-based Algebra I course. The framework presented in this thesis supports the Louisiana State University (LSU) STEM Pathway course entitled Introduction to Computational Thinking (ICT). The course introduces students to a systematic problem-solving approach in which they learn to solve problems computationally, that is, through abstraction, decomposition, and pattern recognition. ICT utilizes the functional programming language Haskell in the educational programming environment “CodeWorld” in order to create pictures and animations.

Jean Piaget, the great child cognitive development psychologist, proclaimed “The goal of intellectual education is not to know how to repeat or retain ready-made truths”; rather, one becomes educated by “learning to master the truth by oneself” (Piaget,
Because of the graphical outputs that one can easily code in CodeWorld, students have the ability to explore an algebraic concept with a computer programmed model, alongside the textbook's given table, equation and graph. This thesis provides additional projects for supporting the Algebra I curriculum through LSU’s ICT course and an overview of the history of computing with an emphasis on highlighting some of the attempts that were undertaken within the past 80 years to use computational thinking and programming to support problem solving across disciplines, including the humanities, math and sciences.
Chapter 1. Introduction

1.1. Introduction

Advancements in technology and the resulting change in society have created a demand for a recalibration of thinking: computational thinking. Computational thinking is a problem solving process, often linked to (but not limited to) ideas in computing, due to its core characteristics of decomposition, pattern recognition, abstraction and algorithm design.

This thesis begins by taking the reader on a journey through time in order to explore the history of the computer. Once one appreciates the evolution of the machine, the educational applications of the computer seamlessly follow.

1.2. The History of the Computer

In order to explore the history of implementing computer programming into the American high school curriculum, one must begin with the short history of the computer. Our timeline of the computer’s history begins around 80 years ago, in 1939. Back then, the early models of the computer were the size of a room, contained zero memory, and in lieu of software relied on physical hardware such as switches and plugboards in order to program the rudimentary machines. Due to the absence of memory, a computer could only carry out one task at once. Consequently, the machine had to be physically reprogrammed and rewired for every new instruction. As a result, the first computers were more or less designed for a single task rather than programmable. This slow procedure is notably illustrated with the British device the
“Bombe”, which was used to help decipher German machine-encrypted messages during World War II, at Bletchley Park, England.

![Figure 1. The Bombe at Bletchley Park, England (National Archives, 2019)](image)

Driven by World War II, engineers, mathematicians, and scientists around the globe worked continuously to create more powerful machines that were smaller in size, faster in speed, replicable, and were able to meet expanded purposes. Innovative computing leaders of the time like Alan Turing and John von Neumann shared a century old dream of building a programmable computing machine. Von Neumann became influenced by Turing’s idea of constructing a universal computing machine with infinite memory after reading Turing’s 1936 paper, *On Computable Numbers* (Turing, 1937). Von Neumann further developed two important concepts that directly affected the computer’s program structure, “shared program technique” and “conditional control transfer” (Von Neumann, 1945). The shared program technique suggested the machine should not need to be hand wired for each new program. The conditional control transfer introduced the concept of control structures within an algorithm which grants blocks of code the ability to be reused, and branch based on conditions being met or not. These concepts could be achieved through building a computer that follows Von
Neumann architecture, a design released by John von Neumann in his 1945 paper, *First Draft of a Report on the EDVAC*¹ (Von Neumann, 1945). In short, von Neumann describes a computer with a central processing unit (CPU) composed of an arithmetic/logic unit in conjunction with a control unit, memory directly accessible by the CPU, and input/output mechanisms. Despite his clear vision, von Neumann didn’t actually build the computer.

In 1975, Professor Simon Lavington explored British computer history in one of the first published computer history books, *A History of Manchester Computers* (Lavington, 1975). The engineering collaboration of Williams and Kilburn are the first major events described in Lavington’s book.

Fredrick Calland Williams (known as F.C. Williams) worked as an electrical engineer at Britain's Ministry of Supply Telecommunications Research Establishment (TRE) during WWII, where he made contributions to the development of Britain’s radar technology. After the war, Williams was invited to travel to the United States for a year to contribute work to the Massachusetts Institute of Technology’s (MIT) 24-volume series on Electrical Engineering. While working there, he learned of researchers’ interest in and attempts to digitally storing data. Upon returning home in 1946, Williams, influenced by the electronic storage attempts in the United States, successfully stored data on a Cathode Ray Tube, which was patented as the Williams Storage Tube. Within days of receiving the patent, Williams left the TRE to accept the position of the Chair of Electrical Engineering at the University of Manchester. He took with him Thomas Kilburn, a member of his TRE team and a graduate of the Cambridge

---

¹ Electronic Discrete Variable Automatic Computer
Mathematics Department. Each of the men brought along their own strength and motivation. Kilburn was a computer design enthusiast while Williams was “enthusiastic about building a small computer to verify the usefulness of his storage invention” (Lavington, p2, 1975). Williams and Kilburn were destined for success as it was observed by the University of Manchester’s Math Department head Max Newman, “There are two distinct fields of inquiry connected with large automatic computing machines. These are first, the engineering problem of designing the machine...secondly, the mathematical and logical problems of finding the best use of such machines and investigating their effect of the development of mathematics itself” (Lavington, p2, 1975). This recognition led the Math Department Head to connect Williams and Kilburn to “Turing’s idea of a stored-programme computer and explained what facilities were necessary for a computer” (Copeland, pg. 209, 2004).

In June 1948, Williams and Kilburn’s collaborative efforts led them to successfully incorporate the William’s Storage Tube into the design of their experimental computer, the “Manchester Baby”, allowing them to digitally store the very first computer program.

Lavington’s 1975 book *A History of Manchester Computers* provides a description of the “Manchester Baby”. According to Lavington, the “Manchester Baby” was the size of half a room and programmed in binary form by using 32 buttons and switches as input devices that set the value of each of the computer's 32 bits to either 0 or 1. With the installation of the William’s Tube, the “Baby” had an internal memory that contained 1 kilobit of memory (on comparison: a 2019 smart phone has up to 16,000,000 kilobits). The William’s Tube is the first example of what is now known as a computer’s Random-Access Memory (RAM). Kilburn wrote the first (and only) program
the computer ran which was an algorithm (using only addition and subtraction) that consisted of 17 instructions which calculated the highest proper factor of a $2^{18} (262,144)$ by trying every integer from $2^{18} - 1$ downward. After 52 minutes, the machine arrived at the correct answer, 131,072. In response to the successful execution, Williams triumphantly exclaimed, “It was a moment to remember. This was in June 1948, and nothing was ever the same again” (Napper, 1999).

Baby’s program was written in 1GL, a first-generation programming language, commonly known as machine language. Machine language is a low-level programming language, characterized by its instructions to store a system’s bits as 1’s and 0’s. It is a raw, primitive language without any abstraction that does not require any compiler (translator) to instruct the machine what to do.

When thinking about programming computers at this time, it is important to remember each machine was independently crafted and unique. Each computer had its
own “machine specific” program that was non-portable to other machines. The programmers were the people who built the machine, giving them the required level of sophistication and knowledge of their computer’s hardware. As the demand for capable programmers increased, however, “the difficulties of explaining machine code to potential programmers who knew nothing about hardware became apparent and a real problem” (Fairhead, 2017). Low level programming languages are an arduous task to write. They are even tougher to read and debug. Another complication was that “the cost of programmers associated with a computer center was usually at least as great as the cost of the computer itself” (Backus, 1978).

With the success of the stored-program concept, the rate of computing innovations finally hit the knee of the exponential curve in the increase in computational advancements.

In 1954, IBM, “the world leader in providing computer systems for both business and scientific applications” (IBM, 1976), released the IBM 650 the world’s first mass produced computer. This computer was designed with the intention of producing a “small, reliable machine offering the versatility of a stored-program computer that could operate within the traditional punched card environment” (IBM, 2003). This computer no longer relied on the flipping of switches to input the program, rather its programmers “wrote” the program on punched cards. Additionally, the machine needed to be “capable of performing arithmetic, storing data, processing instructions and providing suitable read-write speeds at reasonable cost” (IBM, 2003). This system utilized a new kind of storage, a magnetic drum, which resolved speed and storage problems of previous designs. These features proved favorable across industries as IBM ultimately
produced and delivered 2000 computers to various customers, surpassing their originally intended count of 50. The IBM 650 was capable of applications beyond scientific computing such as payroll processing, customer billing and even analysis of flight tests. It was originally programmed in machine language, however, within a few years of its release, and the soon arrival of high-level programming languages, the mass-produced computer was able to be programmed by a larger audience.

In 1956, the Massachusetts Institute of Technology (MIT) experimented with an alternate approach to inputting a program into the computer. Using their Project Whirlwind computer, punched cards and paper tape were replaced with a Flexowriter, an electrically controlled typewriter that allowed its users to enter commands through a keyboard. Despite its practicality, the keyboard was not mainstreamed until about 1975 with the emergence of personal computers.

In 1957, mathematician-turned-computer scientist John Backus and a team of developers from IBM were preparing to program IBM’s next computer, the IBM 704. The team led by Backus recognized the constraints of low-level programming languages and wanted to reduce the time of planning, writing, and debugging the code for scientific problems to be calculated with the 704. In turn, they developed and published the very first high-level programming language, FORTRAN, named as an acronym for FORmula TRANslation. Backus’s inspiration for the language came from him “being lazy”. In 1979, Backus told Think, the IBM employee magazine, “I didn't like writing programs, and so, when I was working on the IBM 701 (an early computer), writing programs for computing missile trajectories, I started work on a programming system to make it easier to write programs” (Bergstein, 2007). Programmers calling
themselves “lazy” as a motivation for managing the complexity of a program using abstractions is still very common today.

FORTRAN used English words to introduce coding structures such as selections (IF ELSE statements), abstractions (function statements), loops (DO), variables, and arithmetic operations which collectively reduced the number of programming statements necessary to operate a machine by a factor of up to 20. Backus and his team highlighted the accomplishment of their goal of making an easier programming language by pointing to a simple event in their 1957 paper, *The FORTRAN Automatic Coding System* (Backus et al., 1957). The case describes a programmer that attended a one-day course on FORTRAN and in turn wrote a program in four hours consisting of 47 FORTRAN statements only to see the output was incorrect and his program contained a bug. Within minutes, he was able to localize his coding error, correct his error, and run his corrected code. The subject in case “estimated that it might have taken three days to code this job by hand, plus an unknown time to debug it” (Backus et al., 1957).

High-level programming languages are unable to be directly read by a computer and so, FORTRAN was developed with its own compiler. A compiler was a computer program that translates the source code written in a high-level language to machine code. The compiler is an important innovation in computing because user-friendly, high-level languages and, in turn, the opportunity to code was now available to anyone with computer access rather than a few exclusive expert programmers. Remember, “prior machine language programs were written to a specific computer, but a FORTRAN
program could be run on any computer with the FORTRAN compiler installed” (Ucar/Comet, 2016).²

Computers at this time were still very rare; institutions typically had one computer that the entire staff used by the method of batch processing. The limiting method of batch processing began with programmers writing their programs on punched cards or paper tape in order to write their program “offline”. Next, the punched cards, typically a large stack, were handed off to a trained operator who ran them through the computer system, who then may have handed the results back to the programmer hours or even days later. One computer programmer remembers this routine as follows: “The machines were behind locked doors, where only guys—and, once in a while, a woman—in white coats were able to access them” (McCracken, 2014). Considering there was only one computer, this method was effective in terms of decreasing the idle time of a computer that would have been while someone was entering the code. However, one can imagine the frustration of debugging a program that was received hours to days later. Programming in FORTRAN utilized this system of batch processing and punched cards until the mid-1970’s in order to write computer programs.

Figure 4. FORTRAN Punched Card (masswerk.at/keypunch/, 2019)

Figure 5. IBM 650 (IBM.com, 2019)

---

² As a result of FORTRAN’s emphasis on arithmetic operations, the language is considered computationally intense. In 2019 the language is still used in fields requiring scientific computation such as large scale climate models, physics and computational chemistry
1.3. Literature Review

In terms of teaching K-12 students computer programming, the early computer models possessed too many constraints and challenges to be of any practical use in teaching children. As technology continued to evolve, the restricting factors of time, space, and accessibility began to diminish. This section of the thesis explores the history of four main educationally motivated programming languages, although there are countless others that could have been chosen.

1.3.1. Introducing Programming to Student Education: BASIC

The accessibility of FORTRAN quickly inspired other high-level programming languages to be developed shortly after, most notable ALGOL and BASIC, The Beginners All-Purpose Symbolic Instructional Code. In 2014, Dartmouth College celebrated the 50th anniversary of BASIC with a series of on-campus public events along with a video Birth of Basic (Dartmouth, 2014) that documented the birth of BASIC and its immediate global impact. The video chronicles John G. Kemeny, the driving force behind the programming language BASIC. Kemeny was a mathematician, computer scientist, and educator who was strongly influenced in his early career working with Von Neumann on the Manhattan Project and later as Albert Einstein’s graduate assistant. By 1955, Kemeny began his twelve-year position as the chairman of Dartmouth’s Department of Mathematics.

---

3 ALGOL is an acronym for Algorithmic Language. ALGOL follows a similar format to FORTRAN and BASIC but differs in its use of code blocks, nested functions and lexical scope.
In the anniversary video, a Dartmouth colleague of Kemeny’s, Stephen Garland, reminisces on Kemeny’s educational gifts and leadership. He recalls Kemeny having a knack for taking the complicated and explaining it in layman’s terms. In the interview, Garland recalls Kemeny once holding a lecture for an audience of Dartmouth alumni with no mathematical background and proving $E=MC^2$ “using just the essential parts in a language the audience could understand and left his audience convinced they could prove the theory as well” (Dartmouth, 2014). The video reiterates Kemeny’s influential vision that “every student on campus should have access to a computer, and any faculty member should be able to use a computer in the classroom whenever appropriate. It was as simple as that” (Dartmouth, 2014). In 1964, Kemeny and colleague Thomas E. Kurtz released their new high-level computing language, BASIC, with the intentions of providing a programming language that was easy to use, powerful, as well as appealing to inexperienced programmers.

BASIC took ideas from FORTRAN such as an emphasis on arithmetic formulas and control statements while following a similar syntactic approach with the use of all capital English words. Also, for the first time in history, the programming language allowed the programmer to number the lines of code which made for cleaner execution of control structures, and served as an easier way to edit lines, insert new lines, or delete lines of code. Below is an excerpts from a BASIC manual which introduces a user to writing a program in BASIC in order to solve a system of equations.
In order to accomplish Kemeny’s vision of every student having access to a computer, something had to be done about the long wait times of executing one’s code and receiving late feedback on one’s batch processing. Kemeny, Kurtz, and a team of undergraduates explored the idea of timesharing, which is “the operation of a computer system by several users for different operations at the same time” (Lexico, 2019). Their original design joined together General Electric’s Datanet 30 with their GE 225 and attached a large number of teletypewriter consoles. This innovation allowed processors to divide up the computing power, giving each user the impression that they each had their own computer. This assembling of interconnected “personal computers” became known as Dartmouth Time-Sharing System. On May 1st, 1964 at 4:07am, the birth of BASIC was marked “with the operating system and compiler working for two different computers at once” (Dartmouth, 2014).

Timesharing was significant because the system allowed up to 30 students to write programs at once and execute them in a reasonable amount of time, opposed to the delays experienced in the batch processing systems. Because of the combination of immediacy and simplicity, Dartmouth was able to establish the largest computing
operation of the time. Roughly 85% of students at Dartmouth made use of the computer, thanks to the Dartmouth Time-Shared System. If a student had an idea, they were able to test it out. Not surprisingly, many of the computing curious students spent their off time creating and programming games such as the three-dimensional tic-tac-toe game Qubic, and even football simulations. Thus, our timeline of computer-gaming begins at Dartmouth, 1964.

In the fall of 1964, Kemeny was on the school board of the nearby Hanover High School district and installed a teletype at the high school. They quickly learned that “high school students were just as eager and just as good as undergraduate students at writing programs” (Dartmouth, 2014). Word spread and soon after they had over 100 users including seven secondary schools in the New England area. The demand on the system became so high the phone company had to add new trunk lines to the town of Hanover. Kurtz estimated that “before Bill Gates got into the [computer] action at all five million people in the world knew how to write programs in BASIC” (Dartmouth, 2014). This unprecedented growth in computing practice was attributed to approximately 80 time-shared systems in the US that offered BASIC as one of its languages.

In 1965-1967, soon after BASIC’s influence spread, the National Council of Teachers of Mathematics (NCTM) conducted a pioneering study that was published in 1972 discussing the integration of the computer in the high school math classroom, titled *Computer-Assisted Problem Solving in School Mathematics* (NCTM, 1972). This study examined the following questions:
1. "Does the activity of writing, executing, and studying the output of computer programs related to problems in the regular mathematics curriculum affect mean student achievement?"

2. Is there a differential effect of computer use on students of varying levels of prior mathematics achievement?

3. Are there curricular areas where the use of the computer particularly contributes to or detracts from mathematics achievement?" (NCTM, 1972)

The study was conducted over two years and had examined comparative groups of 7th graders and comparative groups of 11th graders. All groups received the same math instruction, in accordance with their grade level, and the experimental group of each grade also received lectures and practice in BASIC. In support of the math curriculum, the computer classes wrote and processed computer programs involving the problems, concepts, and skills concurrently with the mathematics course.

The evaluation of the 7th graders consisted of seven posttests that measured the achievement gains of algorithmic math problems and one posttest measuring non-content specific problems in a test titled "thought problems". Results from the seven algorithmic math tests showed no significant difference between the comparative groups two consecutive years in a row. The 11th graders' math achievement followed a similar trend, there was no significant difference between the posttest's achievement scores. The researchers ultimately concluded that "It would appear that while the computer treatments of the studies reported here did not significantly increase skill-oriented achievements, the activity of designing programs and studying computer output did not interfere overall with the development and maintenance of computational or algebraic skills" (NCTM, 1972). The computer programming did not significantly contribute to mathematics achievement, but it did not detract from it either. Because of

---

4 The "Thought Problems Test" consisted of detailed word problems that required analysis, reasoning and insightful solutions. The items had no direct application of 7th grade curriculum or computer programs.
this inconclusive result, there were many educational research questions and reformulations of the NCTM experimental research study left to be explored.

In the 7th grade comparative groups there was, however, a significant difference between the experimental and the control groups’ performance on the more conceptual “Thought Problems Test”, with the experimental group scoring significantly higher. The researchers note this difference as “one of the conjectured benefits from constructing, correcting, and refining computer programs is an improved ability to handle unfamiliar problems” (NCTM, 1972). Perhaps programming in BASIC didn’t improve one’s understanding of content specific standards related to performing mathematical operations, but there is reason to hope that the opportunity for developing one’s computer programming skills help to support general mathematical practices.

Despite its widespread availability, BASIC held a stronger and more relevant reputation amongst the college audience than the K-12 student population. Seymour Papert, a computer scientist and research assistant at MIT, acknowledged this narrow audience and wrote in his 1980’s book Mindstorms, “Programs in BASIC acquire so labyrinthine a structure that in fact only the most motivated and brilliant ("mathematical") children do learn to use it for more than trivial ends” (Mindstorms, 1980). Papert argues that BASIC has too much syntax and is too complicated for a young student to learn.

1.3.2. Programming for the Transfer of Knowledge: LOGO

In 1960, MIT released the Programmed Data Processor-1 (PDP-1) which “marked a radical shift in the philosophy of computer design: it was the first commercial computer that focused on the interaction with the user rather than just the efficient use
of computer cycles” (Computer History Museum, 2019). The PDP-1 was still the size of roughly two large refrigerators, but unlike the analog computers from decades prior could fit comfortably in the corner of a room. The PDP-1 also included many of the standard hardware features found in today’s typical computer such as a keyboard, a cathode ray tube graphic display (CRT) used as an input/output screen, and the unit required no air conditioning. The PDP-1 also had another input device, the lightpen, which was used in conjunction with the CRT to draw and send user input through the screen. These defining features set the stage for the one of the first user interactive video games, Spacewar!, a game in which two players can combat one another in an intergalactic setting where the computer used over 100,000 calculations per second to compute real world physics of the ship’s motion, gravity, user control inputs, and the relative position of the stars and the sun. There were 53 PDP-1’s produced until their end of production in 1969.

In 1967, Seymour Papert and fellow digital learning research scientists of MIT, Wally Feurzeig and Cynthia Solomon, developed an educational programming language for the PDP-1, which was to be implemented in the K-12 classroom. The programming language the team created was titled LOGO\(^5\). LOGO is a “general-purpose computing language that is simple to learn, with a rich and expandable vocabulary that allows the programmer to build procedures, name them, and use them to build further procedures in order to manage a program’s complexity” (Soloman, 1982).

LOGO is an educational programming language with an emphasis on the programming commands and output display rather than the syntax of the code. The

---

\(^5\) LOGO is not an acronym, rather is derived from the Greek \textit{logos} meaning thought.
language is intended for children as the written code results in a visually appealing geometrical/graphical output. LOGO’s defining feature is the “turtle”, which began as an on-screen retractable pen. This turtle acts as a cursor in which a command of PENDOWN followed by FORWARD 100, would leave a visible trace of the path as the turtle steps 100 turtle paces forward. Other predefined commands include RIGHT(degree) and LEFT(degree), which direct the turtle to rotate an instructed degree turn. By combining these drawing commands, a child can program their turtle to draw a shape as simple as a line to complex mathematical roulette curves. This wide range of opportunities is known as a “low floor, high ceiling” approach to create interest and self-investment in computer programming environments.

In addition to the simple commands, a student can expand a process, or drawing, by defining new commands that can in turn define even more commands. Papert cites an example of coding a house, by building by a square and a triangle. Neither a square or triangle is predefined, but a student had the ability to define a square as

SQUARE:
PENDOWN
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
PENDUP

By defining a triangle similarly, one is able to construct a house in a simplified number of commands, see Figure 1. By applying a series of rotations, students could also create more evolved abstract effects. For example, once the SQUARE command is
defined, commanding SQUARE and instructing the turtle to rotate by 36 degrees ten times, gives the picture in Figure 2:

![Figure 8. Code for House (Papert, 1980)](image8)
![Figure 9. Rotations Code (Papert, 1980)](image9)
![Figure 10. PDP-1 (History-Computer.com, 1980)](image10)

The LOGO programming language is more than just a language. In Seymour Papert’s 1980 book *Mindstorms*, he emphasizes that the LOGO Environment is to be implemented with a discovery-learning and constructivist approach, where the students learn from exploration and should have minimal teacher instruction. Thus, “the child, even at preschool ages, is in control: the child programs the computer. And in teaching the computer how to think, children embark on an exploration about how they themselves think” (*Mindstorms*, 1980).

LOGO was created with the intention of strengthening a child’s cognitive ability. According to the Laboratory of Comparative Human Cognition, the strengthening of a child’s cognitive ability can be recognized by an observer as “transfer of knowledge, or, rigorous thinking in other disciplines” (Flavell, 1994). Specific to children engaged in the LOGO environment, Papert’s partner in development, Wally Feurzeig, writes in his 1981 paper *Microcomputers in Education*, seven general
programming concepts that may transfer to the child’s ability to do related tasks in mathematics. They are:

1. the precision of algorithms leads to precision of expressions
2. the use of variables, definitions and functions lead to a natural understanding of general mathematical concepts
3. the heuristics gained from pre-planning and planning-in-action leading to solving any general problem through breaking it down, relating the problem to a previously solved problem
4. debugging leading to finding errors in mathematical work
5. abstracting programs leading to finding patterns in mathematics to accomplish a larger task
6. discussion of programming leading to elevated literacy and language of problem solving in mathematics
7. there is more than one way to accomplish a task (Feurzeig, 1981).

These grand claims of encapsulating the development of mathematical thinking skills grabbed the attention of child psychologist Roy D. Pea. Pea studied child language and cognitive development in the 1970's when his research interests were attracted to understanding how innovations in computing and communication technologies can influence students' learning, thinking, collaboration, and educational systems in general.

In 1981, Pea challenged the claim that a spontaneous transfer of cognitive skills occurs when a student is engaged in the LOGO environment by asking whether or not “learning to program a computer provide such experiences in higher mental functioning that the rigors of computer programming transfer to thinking and problem solving in other areas” (Pea and Kurland, 1984). Roy D. Pea, in partnership with cognitive scientist D. Midian Kurland, released a report titled *LOGO and the Development of Thinking Skills* (Pea, Kurland, & Hawkins, 1985) which described a series of experiments that would continue for years.
Pea and Kurland began their research of students in the LOGO environment research by zeroing in on the claim of LOGO that it strengthens a student's cognitive abilities (e.g. planning tasks), and asked “Does the effectiveness of planning become more apparent to a person learning to program? Does the development of planning skills for more general use as thinking tools become more likely when one learns to program” (Pea and Kurland, 1984)? The cognitive researchers set up an experiment involving four groups of middle school students, an older experimental group, a younger experimental group, and a control group for each. The experimental group participated in weekly LOGO programming activities for an entire school year which were guided by a teacher who had undergone intensive LOGO training the summer before. The control group received no computer programming opportunities. The purpose of this study was to investigate how students who had been doing computer programming for a school year differ in their use of planning skills from students without programming experience.

The procedure of this study followed Papert’s proposed LOGO Environment; the experimental group began the year learning the basics of LOGO in a large group session, followed by self-exploratory usage as “students were encouraged to create and develop their own computer programming projects” (Pea and Kurland, 1984). The study concluded with an experiment which consisted of a task in which the students had to create a map of the most effective classroom chore route.
Each student was evaluated individually and could make as many plans as needed, revising each plan after receiving immediate instructor feedback. The children were instructed to think out loud while planning and were encouraged to use pencil and paper to take notes. This task had to be completed twice throughout the school year. Pea and Kurland considered three different aspects of the students’ plans to measure the potential heuristic gains: efficiency, revisions, and formulation of the chore route. The results were consistent. The older group outperformed the younger group in all three aspects, however there was no significant difference between the experimental and control groups performance in all three categories or in either age group. Pea summarizes: “On the face of it, these results suggest that a school year of LOGO programming did not have a measurable influence on the planning abilities of these students” (Pea and Kurland, 1984).

Pea and Kurland’s results of this initial study on how to create the most effective classroom chore route did not match LOGO’s claims, so the researchers refined and redesigned the experiment, structuring a second study (1982-1983) with an evaluative
task that closer resembled the structural aspects of programming. The in-class LOGO lessons also included more direct teacher instruction, while still encouraging students to create their own projects. While the pre-assessment was identical to the first study, the post assessment was computer-based, which included immediate feedback to the students, similar to the instant feedback in programming, allowing them to monitor their own progress and potentially revise their plans, reflecting a debugging process. This study analyzed the results with respect to three different rubrics:

1.3.2.7.1. Reasoning, Math and Planning Skills of Pre vs Post tests
1.3.2.7.2. Correlation of Math and Reasoning with Programming of Pre vs Posttests
1.3.2.7.3. Programming Ability in the Experimental Group of Pre vs Posttests.

Again, the researchers concluded that “programming experience (as opposed to expertise) does not appear to transfer to other domains which share analogous formal properties” (Kurland, Pea, Clement, Mawby, 1986).

This is not to say at all that LOGO was an unsuccessful project. Many different studies in the early 1980’s had shown significant increases in students’ collaborative abilities when working on projects with computers, compared with their interactions when working on projects without computers. LOGO also introduced children to computer programming early on, and encouraged universities and other research organizations to create their own programming languages and computing literacy programs for K-12 students.

1.3.3. Programming for Technological Fluency: Scratch

In 1993, a team led by Mitch Resnick, a doctoral student of Seymour Papert at MIT, established the first Computer Clubhouse, an after-school technology learning
center which served youth from economically disadvantaged communities. The goal of the Computer Clubhouses was “for participants to learn to express themselves fluently with new technology” (Resnick, Rusk, Cooke, 1998). Papert’s constructivist approach to teaching was clearly imprinted on Resnick since the Clubhouses “provided its participants leading-edge software to create their own artwork, animations, simulations, multimedia presentations, virtual worlds, musical creations, web sites, and robotic constructions in order to allow the students to create and flourish in the technology, not just consume it” (Resnick, Rusk, Cooke, 1998).

Of the myriad opportunities available to students at the Clubhouses, the most germane is the opportunity for robotic constructions. The origin of the robotics program dates back to an eighties-era collaboration with LEGO. By 1980, the LEGO company, famous for its “interconnecting bricks containing the characteristics of development, imagination, and creativity” (LEGO, 2012) had grown into a worldwide toy phenomenon. In 1980, LEGO further widened its consumer base and company purpose by founding its Educational Products Department. This department had —and in 2019 still does— the goal of connecting teachers and educational specialists to LEGO products that would inspire hands-on learning through play and investigation.

In 1985, Kjeld Kirk Kristiansen, the CEO of the LEGO Group, became enthralled with Papert’s 1980 book Mindstorms agreeing with Papert’s constructivist approach to learning. In efforts to collaborate and testing his pedagogical ideas, while also being driven by the desire of wanting the LEGO brand “to be the strongest in the world among families with children” (LEGO History, 2012), Kristiansen reached out to Papert and the MIT research lab. As a result, Papert and his doctoral student Mitch Resnick, teamed
up with the LEGO Group and MIT labs in order to conduct research that integrated the
LOGO programming language with the tangible LEGO blocks in order to bring to life new toys and ideas.

The first project the collaboration tested was the LEGO/LOGO project in 1988. This project was implemented in an elementary school classroom in Boston, with the goal of “getting students to explore design and engineering ideas that are rarely addressed in school by creating behavioral constructs for their LEGO creations” (Resnick & Papert, 1988). In the creations, the children started by building machines out of the traditional LEGO pieces. Then, newer input/output pieces such as gears, motors, and sensors were introduced in order to connect their machines to a computer. Finally, the students wrote computer programs (using a modified version of LOGO) to control the LEGO machines. With the contemporary parts and the programming to bring them to life, the students were able to bring to life the binary operations of their constructs such as turning the lights of a house on and off or opening a home’s garage door when a car approaches. Although there were no quantitative outcomes recorded, the educational values of the LEGO/LOGO project could be seen by the students wanting to continue working during lunch and after school in order to nurture their curiosity and perfect their design. These high levels of motivation lead to the “appropriation of new ideas about science and design” (Resnick & Papert, 1988).

The LEGO/LOGO project had its physical limitations however, as students could only program one computer, independent of others at a time. Also, there were too many wires with Resnick observing that “It is difficult to think of a machine as an autonomous creature if it is attached by umbilical cord to a computer” (Resnick, Martin
R, Silverman, 1996). In 1990, the project evolved into the Programmable Brick. The “Brick” was upgraded from the previous project with goals of “multiplicity”. These goals were accomplished as the Brick could control multiple outputs and check multiple sensors at once. The Brick also had the ability to share data with other Bricks around it. And finally, the Bricks were wireless. Students would write programs on a computer, then using a wire, upload the program to the Brick where it could be stored with multiple other programs that could be executed through using a knob on the brick to scroll through a menu of uploaded programs. In his 1993 report “Behavior Construction Kits”, Mitch Resnick explains “With Programmable Bricks, children can spread computation throughout their world, making familiar objects responsive and interactive. We view Programmable Bricks as a way of making ubiquitous computing accessible to children” (Resnick, 1993).

These smaller, lab-based initiatives led to the public release of the LEGO Mindstorms product. The Mindstorms product may be best remembered by its 1998 user-interactive Robotics Invention Kit that went on sale for $200 just before the holidays. Since its 1998 release, LEGO Robotics has evolved and in 2019, the product is in its third incarnation, LEGO MINDSTORMS EV3. Unfortunately, the educational, inspiring, and engaging product comes with a price tag of $349.95, which limits its accessibility. The price tag was not the only concern, both LEGO and Media Lab researchers had remarked that many school teachers were not using the toy as intended, “they are using it as an ‘instructivist’ toy, rather than a ‘constructivist’ one” (Mikhak, 2000).
Mitch Resnick had collected a lot of research and observations through his partnership with the LEGO Group and was able to continue to work towards his goal of allowing his students at the Clubhouses to be creators, and not just consumers of technology. Resnick was further empowered by the increase in personal home computers and the cries for technological literacy that were getting national attention, demonstrated by the National Academy for Engineering, proclaiming in 2002 that “most people are poorly equipped to recognize, let alone ponder or address, the challenges technology poses or the problems it could solve” (NAE, 2002).

By this time, Resnick had a Ph.D. in Computer Science for a little over a decade and had been studying youth engagement in various disciplines within the Clubhouses. The participants had a strong interest in computer-based programs such as Photoshop, and other graphic and video manipulation software, however, the computer programming opportunities were often too difficult, too irrelevant, and had a limited social image. Assisted by the research team of John Maeda of MIT and Yasmin Kafai of UCLA, Resnick and this team set out to create a new programming environment with the intention of strengthening the “development of technological fluency in informal-learning settings” (Resnick, Kafai, Maeda, 2003). This project birthed the programming environment called Scratch.

Scratch has been a powerful and innovative programming language from its origin since the language was created by the organization behind the Clubhouses that had intended to use it for their program’s participants. The design and development of Scratch was also guided and revised, at every step in the process, by the needs and constraints of Computer Clubhouses, meaning the language is relevant to the fiscal and
emotional needs and abilities of the at-risk youth at hand. Scratch was exclusive to the Computer Clubhouses from 2003-2007 before MIT launched the language and environment to the public. Resnick and his team used their previous experiences with the LEGO collaboration to guide their creation. Resnick reflects:

“We wanted to develop an approach to programming that would appeal to people who hadn’t previously imagined themselves as programmers. We wanted to make it easy for everyone, of all ages, backgrounds, and interests, to program their own interactive stories, games, animations, and simulations, and share their creations with one another” (Resnick, et. al, 2009).

The LEGO influence can be seen within the Scratch language as the code’s syntax is composed of “programming blocks” that snap together only in ways that make conventional programming sense. These virtual blocks are used to create programs with an emphasis on diversity and personalization in order to engage and allow a child to program a range of programs from purposes of entertainment, such as games like “tag” to purposes of enriching a concept, such as creating a story about the journey to the center of the earth that could support an earth science curriculum. This diverse output of programming animated characters, known as sprites, is the result of Scratch’s visual programming environment.
Scratch was successful in achieving its goal of introducing computer programming to a broader audience. At the beginning of 2019, Mitch Resnick announced that “there are more than 30 million registered members on the Scratch website, and every month 1 million new people join (most of them ages 8-16). Every day, Scratch community members create more than 200,000 new stories, games, animations and other projects on the site” (Resnick, 2019). The Scratch block-based approach to coding has become the go-to standard for introducing children to coding, with endorsed Scratch dominated curricula for the College Board’s AP Computer Science Principles course.

In 2015, Dr. Michal Armoni and colleagues of the Weizmann Institute of Science published a study titled From Scratch to “Real” Programming (Armoni et al., 2015) which examined the transitions between middle school students learning Scratch and their attitude and ability to learn a text-based programming language in high school such as C# or Java. The study included “120 students, studying in five different classes, in four different schools, taught by four different teachers. Two of the classes consisted of students who had not previously taken a CS course in school. Three of the classes were mixed: some students had not taken a course in CS, whereas other students had studied a course on CS concepts with Scratch during the previous year” (Armoni et al., 2015). The experimental group consisted of 44 students who had been exposed to Scratch the previous school year while the control group of 76 students had not. The researchers posed the following question: “Does learning Scratch in middle schools lead to improved learning of CS at the secondary level? If so, how does it contribute to improved learning?” (Armoni et al., 2015)
These research questions were explored through a quantitative approach in the form of six conceptual checks, an interim test and a final exam, which assessed content taught in the 10th grade course. The researchers also took a qualitative approach by conducting interviews and observations. In order to focus on the overlap of concepts of Scratch and the text-based languages, the study was conducted from the beginning of the school year until the end of January. Quantitatively, the experimental group performed significantly higher on conceptual checks, however as the course went on, the scores balanced out there was no significant difference between the groups on the interim test and final exam. There was, however, a significant difference between the experimental and control group’s creativity score on the final exam and qualitative measures showed confidence and familiarity with the overlapping concepts. The study recalls the experimental group recognizing programming concepts such as variables and conditional statements in the new language without any teacher explanation. An interviewed teacher within the study also cites increased student learning as she observed that “I felt I could teach the material faster, compared to previous years” (Armoni et al., 2015). And finally, the study reports a doubling of enrollment in the secondary school computer science course.

Within the past decade, there have been multiple visual-programming languages released by top tier research universities, with the goal of introducing a broader range of young people to computer coding and practicing technological fluency. For example, Carnegie Mellon released their own programming language, Alice, a drag and drop language that allows the user to create 3D animated stories and games. Similarly, Microsoft launched its own visual-programming game design language, Kodu. Kings
College of London has the Java based language Greenfoot; and the list goes on. While these languages each have their limitations, their purpose is all the same, namely to provide an engaging introduction to computer programming for children.

1.3.4. Programming for Algebra I Enrichment: Bootstrap

After decades of research, rebuttals, and initiatives of introducing programming into the K-12 classroom, a group of researchers, funded by the National Science Foundation and Brown University, created the Bootstrap Curriculum with the subtitle “Integrating Computing for Algebra, Physics and Data Science for all Students” (Bootstrapworld, 2019). The Bootstrap curriculum is unique because it heavily supports algebra concepts. The course is roughly 17 hours long and boasts its diverse applications and claims that “Our introductory class can be integrated into a standalone CS or mainstream math class, and aligns with national and state math standards” (Bootstrapworld, 2019).

Like Pea and Kurland, the Brown researchers asked the question, “can we identify specific problem-solving practices in computing that have direct analogs to processes in mathematics?” Furthermore, “can we teach them in such a way that students realize performance gains in mathematics” (Schanzer, 2015)?

The Bootstrap curriculum plays on children’s excitement over video games. The premise of the curriculum is that a student will build his own simple video game using Bootstrap’s own text-based language. Such student created examples include catching candy, avoiding the lunch lady, meeting aliens, etc. In each unit the student will be introduced to a new game feature, programming concept, and math concept at once. Programming concepts include expressions, circles of evaluations, string and image
operations, defining functions, Boolean operators, and conditionals. The included math concepts are coordinates, domain and range, kinds of data, functions as formulas and tables, inequalities, piecewise functions and the Pythagorean Theorem. The final unit of the curriculum is polishing the game for presentation, which focuses on code review and explaining math concepts to others. See the below table.

<table>
<thead>
<tr>
<th>Unit</th>
<th>Game Feature</th>
<th>Programming Concept</th>
<th>Math Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Locating elements on screen</td>
<td>Expressions, Circles of Evaluation</td>
<td>coordinates</td>
</tr>
<tr>
<td>2</td>
<td>Creating text and images</td>
<td>String and image operations</td>
<td>Domain, range, kinds of data</td>
</tr>
<tr>
<td>3-5</td>
<td>Making moving images</td>
<td>Defining functions, examples</td>
<td>Multiple function representations as formulas and tables</td>
</tr>
<tr>
<td>6</td>
<td>Determine when game elements are off-screen</td>
<td>Booleans and Boolean operators</td>
<td>inequalities</td>
</tr>
<tr>
<td>7</td>
<td>Responding to key-presses</td>
<td>conditional</td>
<td>Piecewise function</td>
</tr>
<tr>
<td>8</td>
<td>Collision detection</td>
<td>(nothing new)</td>
<td>Pythagorean Theorem</td>
</tr>
<tr>
<td>9</td>
<td>Polishing games for presentation</td>
<td>Code reviews</td>
<td>Explaining math concepts to others</td>
</tr>
</tbody>
</table>

Figure 14. Curriculum Structure (Schanzer, 2015)

Different from the past programming initiatives, the Bootstrap researchers tested highly concentrated mathematical standards. They hypothesized,

1. Students who complete Bootstrap will improve in their performance on algebra word problems and function composition problems.
2. Students who complete Bootstrap will show more improvement in performance on algebra word problems and function composition problems than students who did not take Bootstrap. (Schanzer et al., 2015)

The hypotheses were assessed with pre- and posttests composed of questions taken straight from the Massachusetts state assessments in mathematics. There were eight word problems and nine function composition questions on each of the pre- and posttests. The tested students were a group from Massachusetts, with a comparison group in Florida and a comparison group in Illinois. The results showed a strong change in the pre- and posttest knowledge in all three of the experimental groups.
The experiments investigated narrow concepts as the researchers omitted testing questions on domain and range, input and output because the Massachusetts tests did not include questions on such. While Bootstrap has proven an improvement in the student’s ability to answer word problems, the curriculum is only 17 hours long and therefore has a very basic coverage of math and programming concepts. The rigid course structure also does not allow for much room for differentiation of student skills.

1.4. Advanced Placement Computer Science Principles

In 2008, the National Science Foundation and the College Board teamed up in order to create a new Advanced Placement (AP) computing course, “Computer Science: Principles”, referred to as APCSP. This course was designed to “introduce students to the central ideas of computing and computer science, to instill ideas and practices of computational thinking, and to have students engage in activities that show how computing and computer science change the world” (Astrachan et. al, 2011). Due to its introductory nature, the course was also designed for a diverse population of students that may typically not take an Advanced Placement or computing course. Unlike the College Board’s other computing course, AP Computer Science: A, which strictly focuses on programming in the text-based language Java, APCSP focuses on introducing students to a broad range of ideas and concepts in computing, with no prior coding experience necessary. The key ideas include, but are not limited to, creativity, abstraction, programming, and global impact.

The APCSP course presents the unique opportunity to the instructor of teaching any programming language since coding is just a small aspect of the course. Many universities and tech companies have joined the APCSP movement and created
curricula an instructor may choose from that are officially endorsed by the College Board. This list grows each year but notable participants include Apple teaching their language Swift, Harvard University teaching coding elements from Scratch, C, PHP Python, and Code.org\(^6\) teaching students to make apps through their own JavaScript-based “App Lab”.

AP Computer Science Principles was first launched in 2016-2017 school with nationally over 50,000 students taking the APCSP exam. In just one year, over 70,000 students had taken the exam. The APCSP exam is different from every other AP course; the exam is broken into two sections, performance tasks and the end-of-course multiple choice AP Exam. There are two performance tasks, the Create\(^7\) and the Explore\(^8\) tasks which in combination account for 40% of one’s final AP score. These tasks are completed during class, without assistance from the teacher. Because of these open-ended assessments, the versatile course again takes a low floor, high ceiling approach where students will end up taking just as much out of the course as they put into it. The College Board created an introductory hype video, highlighting their programming for all mission with a small rural town teacher expressing in the video that “The best thing about teaching AP CSP is watching students grow in their self-confidence and in their problem-solving abilities” (College Board, 2018).

\(^{6}\) Known best for their “Hour of Code” challenges and initiatives
\(^{7}\) Students will develop a computer program of their choice (College Board, 2019)
\(^{8}\) Students will identify a computing innovation, explore its impact, and create a related digital artifact (College Board, 2019)
1.5. Pearson’s MyMathLab

Using technology to enrich a K-12 math curriculum is not a new pedagogical area of inquiry. In 2012, middle school math teacher, and Louisiana State University’s Master of Natural Sciences graduate, Sarah Dyer writes of a “dual-intensity” approach to improve her 7th graders math achievement. Dyer tested the use of Pearson’s MyMathLab in her classroom and after her students showed unprecedented gains she writes, “working on the computer gave my students a sense of autonomy and pride in their class environment that they did not have before we had access to the program” (Dyer, 2012). She believes the use of MyMathLab enriched the procedural fluency of students. In order to balance conceptual understanding, the second part of her “dual-intensity” scheme was implemented through extended written tasks within the classroom. Dyer acknowledges that “computers cannot and should not replace teaching towards understanding, but can and should be used to promote and foster those understandings and intuitions” (Dyer, 2012).

1.6. LSU STEM Pathways Project

In 2006, the head of the Computer Science Department at Carnegie Mellon University Jeannette Wing, published an article promoting computational thinking as an approach to teaching computer science. She argues that all students should learn and practice computational thinking. She write that, “Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science” (Wing, 2006). This definition can
best be exemplified by when having to solve a particular problem, asking oneself “How difficult is it to solve? and What’s the best way to solve it” (Wing, 2006)?

In 2015, the East Baton Rouge Parish School System (EBRPSS) rebuilt the low-performing Lee High School with the intention of rebranding the school as a STEM based magnet school. The structure of a student’s course work at the new STEM school would consist of students earning dual enrollment and advanced placement in their academic core classes and engaging in project-based learning opportunities supporting a self-chosen elective domain such as Pre-Engineering, Digital Design and Emergent Media, Biomedical Sciences or Computing. Upon choosing an elective domain, the student is provided with a pool of four core classes and a wide range of elective classes in the chosen pathway. Seeking curriculum support for the elective pathway programs, EBRPSS teamed up with the Gordon A. Cain Center for STEM Literacy at Louisiana State University. Together, this comprehensive partnership created rigorous STEM curricula for the four domains and provided middle school and high school teachers with advanced professional development programs in the fields of Pre-Engineering, Digital Media, Computing, and Biomedical Sciences in order teach the elective courses. These curricula and domains are supported by Louisiana Department of Education’s STEM Initiative, and are known as the LSU STEM Certification Pathways.

Each of the four pathways have four required core classes and a list of additional elective courses students may take in order to earn a Silver STEM Diploma Seal or Gold STEM Diploma Seal, respectively⁹. When a school offers more than one pathway,

---

⁹ The STEM endorsed certificates earn students’ consideration for LSU scholarships and participating schools receive substantial career technical education funds for offering the LSU Pathway courses.
a student must decide early in their high school career on the domain they would like to fill their elective credits with. Such a restricting decision is difficult for a young child to stick with and so the transfer of pathways was considered in the pathways’ development stages. It was decided that “Regardless of the academy a student chooses, they will be introduced to programming and coding” (Begat, 2016). Hence, the Introduction to Computational Thinking (ICT) course was created and integrated as a requirement into all four pathways. ICT uses a functional programming language to code graphics, and the goal of the course is to teach students fundamental computational thinking skills, such as abstraction, decomposition, and pattern recognition. These skills are transferable to the other pathways as well as courses outside of the pathways.

The LSU Pathways project continues to grow; in 2019, there are 2,800 students in 40 middle and high schools in 20 Louisiana school districts taking LSU Pathway courses.

1.7. An Overview of the LSU Introduction to Computational Thinking Course

Introduction to Computational Thinking (ICT) is the first recommended course in the LSU Computing Pathway and should be taken as the first or second course in the other three pathways (Biomedical Sciences, Digital Design and Emergent Media, Pre-Engineering). The course is being designed by LSU Research Associate, Fernando Alegre. Its objectives are to introduce students to basic ideas of computational thinking and building their self-confidence in visualizing and modeling STEM content. The computing language taught in the course is a simplified version of Haskell, a functional programming language that has strong similarities to concepts and notations from
algebra. In the ICT course, the Haskell coding happens in a web-based educational
computer programming environment called CodeWorld\(^\text{10}\). CodeWorld is a fun and
engaging environment for young students as CodeWorld allows one to begin by “coding
drawings, animations, and even single-player and multi-player video games using
mathematics, using shapes, colors and geometric transformations” (CodeWorld, 2019).

The CodeWorld environment contains an editing panel, an output panel and a
console panel which provides a space for CodeWorld to display error and warning
messages to the programmer. Along with the ability to program graphical outputs,
CodeWorld introduces the user to other computing elements such as lists, functions,
random numbers, data manipulation, recursion, and conditional functions.

\(^\text{10}\) CodeWorld was created by Chris Smith, a senior software engineer at Google with the initial design for
young students to learn mathematics and computer science concepts. Fernando Alegre is a core
contributor to CodeWorld and continues to collaborate with Smith with research and technical support of
the CodeWorld platform and LSU Computing Pathway
Any teacher interested in teaching the course has to undergo extensive teacher training the summer prior to teaching the course, no prior coding experience with programming is necessary. The ICT curriculum currently contains five units, and within each unit there are seven lessons. Each lesson begins with a short lecture, introducing a new function or other computing element, followed by guided activities, such as draw a blue rectangle layered on top of a red circle. Each unit concludes with an open-ended project that states a comprehensive goal using the skills from the unit, such as draw a quilt pattern, accompanied by a rubric, which guides the students’ computational thinking process while allowing their creativity to flourish. See Appendix A or https://blackeyepeas.phys.lsu.edu/Review/RPP-2019/ICT/ for the full course outline.

The course implements the idea of a “low floor, high ceiling” which has been guiding principles for the creation of programming environments for children since the early days of LOGO. It essentially means that it should be “easy for a beginner to cross the threshold to create working programs (low floor), but the tool should also be powerful and extensive enough to satisfy the needs of advanced programmers (high ceiling)” (Grover, Shuchi & Pea, 2013).

A main difference between the languages used in the CodeWorld and LOGO environments are the programming paradigms they each follow. LOGO is a procedural language which is “based on sequences of commands that instruct the computer to modify the current state of memory, and dynamically replacing the values that variables represent with new computed values” (Alegre, Moreno, 2015). Students, especially 9th graders, have never been exposed to a variable meaning anything other than a symbol used to represent a single unknown quantity (i.e., suppose \( x + 3 = 5 \), then \( x \) must
represent 2). In contrast, CodeWorld used Haskell which is a functional programming language that allows students to express code directly as algebraic functions, hence linking the student’s programming understanding to familiar math concepts.

CodeWorld is different than Scratch in many ways with the most obvious being the way a user codes the program, a text-based syntactical approach versus a block syntactical approach. Again, CodeWorld follows a functional approach where the user uses mathematics in order to create one’s output whereas Scratch follows a procedural approach using small scripts consisting of loops, conditionals and function calls in order to control the flow of the program.

And finally, ICT takes a different approach to complement Algebra I and Geometry than the Bootstrap curriculum due to the length of each course; ICT is a one-year course, Bootstrap is only 17 hours.

The Introduction to Computational Thinking course is a fun course. The students love it. In fact, in my current APCSP course where we are programming in Scratch, the students who took ICT last year are often heard chanting “CODEWORLD, CODEWORLD”, in hopes of returning to the environment.

My thesis project takes advantage of the LSU Introduction to Computational Thinking course by linking it to a topic relevant to all high school math teachers, Algebra I. The thesis provides a framework for using ICT in order to support the Algebra I curriculum. I have designed different projects to be embedded into particular units or standalone projects that integrate Algebra I Standards. These projects have differentiated activities within them, with the intentions of providing accessibility to the
lowest to the highest performing students. These projects will be explained in Chapter 3 and can be found in Appendix C.

As a side goal, I have taken inspiration from Kemeny, Papert, and Resnick to encourage more student engagement in computer literacy.
Chapter 2. Supporting Algebra I through the Introduction of Computational Thinking Course

2.1. An Overview of the Additional Content

The Introduction of Computational Thinking (ICT) course is a yearlong course that is currently being offered to students ranging from 8th grade to 12th grade, with the majority of the students taking the course as a 9th grader. It has been strongly advised that the students have successfully completed or are concurrently enrolled in Algebra I, but the requirement is not absolutely necessary. Students need no prior coding experience for this course.

The course is broken into two semesters, with the first semester providing fundamental concepts and syntax and the second semester applying the skills learned in more detail and more depth. The first semester of the course is titled “Using Code in Order to Represent Objects and their Relations”. Some example objects include a triangle, a five pointed star, and a quilt made out of basic shape repetitions.

At the core of each unit, students must understand and apply geometrical concepts such as the defining arguments of a circle and rectangle and be able to apply transformations such as scalings, translations, and rotations to these shapes. These geometrical concepts are applied in a coordinate plane which is an 8th grade geometry standard (8.G.A.3). In its standard library, CodeWorld utilizes a 20x20 Cartesian coordinate plane and so all students should be familiar with plotting an ordered pair.

\[\text{8.G.A.3: Describe the effect of dilations, translations, rotations, and reflections on two-dimensional figures using coordinates. (Rotations are only about the origin, dilations only use the origin as the center of dilation, and reflections are only over the y-axis and x-axis in grade 8.)}\]
In terms of programming, students learn to create and use global and local variables, lists, and unevaluated expressions. Through these math and computing concepts, the students engage in the software development process in order to code graphical mini-masterpieces which require them to plan, design, implement, test, review and repeat.

The second semester of the ICT course is titled “Applying Computational Thinking to Solve Problems” and demands a higher complexity of skills. Some example problems include drawing a clock with accurate measurements, applying transformations in the correct order to intentionally preserve or manipulate an object’s properties, or extracting data from a table in order to calculate discounts and sales taxes of a fictional sock company. In order to manage the complexity of the activities, students learn all about creating and using functions with multiple parameters. These CodeWorld functions use the “function notation” of mathematics which is introduced for the first time in the Common Core State Standards in Algebra I (F.IF.A). The ICT curriculum continues to gain speed as more abstract concepts are introduced to the students such as using random numbers, experimenting with the simple user interface, and using lists to store homogeneous data or tuples to store heterogeneous data collections.

All of the skills covered in the course lead up to a final unit introducing the tools needed to create animations. Animations are different than drawings in CodeWorld

---

12 6.NS.C.6: Extend number line diagrams and coordinate axes familiar from previous grades to represent points on the line and in the plane with negative number coordinates.
13 A1: F-IF.A: Understand the concept of a function and use function notation.
14 Example of homogenous data storage: Calories=[98, 78, 100, 121]
15 Example of heterogeneous data storage: Identity=(hairColor, name, age)
because they take in a function with the argument of time and enable the user to animate a picture. The argument of time is a changing number that represents the time that has elapsed since the program started. This ever changing argument significantly raises the ceiling of imagination and creativity as one can now create anything from bouncing balls to the infinite zoom into a fractal.

As the course is currently structured, it is an activity packed, discovery and project based course that teaches students programming and computational thinking skills that are transferable to other programming languages and ultimately problem solving in STEM fields. The course also provides a framework to integrate and enrich math concepts from the Common Core State Standards. In a slightly refined form, these nationally recognized standards have been adopted by the state as the Louisiana State Standards for Mathematics.

I have dived into the current ICT course along with Louisiana’s 8th grade and Algebra I standards in order to create more ICT lessons with the goal of supporting the 9th grade Algebra I curriculum. These supplemental lessons are:

1. Embedded Content: Dilations
2. Embedded Content: Rotations
3. Embedded Content: Scalings
4. Embedded Content: Function Notation
5. Linear VS. Exponential Functions Project
6. Systems of Equations Project

Each lesson uses CodeWorld in order to model, visualize, and explain a mathematical concept in a way a standard Algebra textbook is unable to. The lessons begin with an interactive model which allows students to make observations and guide them to mastering the mathematical truth by themselves. Each lesson concludes with application problems of each related standard, taken from Louisiana’s formative
assessment system, LEAP 360. And finally, each lesson includes extension questions for the advanced and curious student (high ceiling).

The lessons are divided into two distinguished categories (Embedded Projects and Additional Projects) that differ in terms of execution throughout the ICT curriculum.

2.2. Embedded Projects

The following lessons, see also Appendix C for more detail, are titled embedded projects because they are supplemental activities that should be embedded into lessons that are already in the ICT curriculum. These lessons allow the students to explore the concept with a user-interactive model, followed by guiding questions that are designed to help students develop conceptual understandings and ultimately apply the skill to their programs in CodeWorld.

2.2.1. Embedded Project: Dilations

The first supplemental lesson, dilations, connects the function \texttt{dilated()}\textsuperscript{16} to the 8th grade geometry standard 8.G.A.3\textsuperscript{17}. This function is introduced in Unit 2, Lesson 2, titled “Dilations and Rotations” which is in the beginning of the first semester therefore, students are more likely to connect the dilation concept in CodeWorld to their 8th grade knowledge. In 8th grade, images are dilated from the origin of a Cartesian coordinate plane, just as they are in CodeWorld, as long as no other transformations have been applied.

\textsuperscript{16} System recognized functions light up bold and blue in the CodeWorld environment
\textsuperscript{17} 8.G.A.3 Describe the effect of dilations, translations, rotations, and reflections on two-dimensional figures using coordinates. (Rotations are only about the origin, dilations only use the origin as the center of dilation, and reflections are only over the y-axis and x-axis in Grade 8.)
Teachers should the teach the supplemental dilation lesson after the students are introduced to the new `dilated()` function and before the students move onto the second part of the ICT lesson on rotations.

In the supplemental lesson, students are given a model in CodeWorld and are able to modify the code in order to investigate such properties of dilations as vertex placement, side lengths, and angle measurements between a preimage and image. After constructing conclusions regarding the properties of dilations, students will answer questions similar to those assessed on LEAP 360, and will check their calculated answers by modeling the scenarios with CodeWorld. The lesson concludes with the students making predictions involving dilations of a circle. This concept is not in the math standards, but mastery of the dilation concept with CodeWorld’s elementary shapes will be helpful in coding graphical masterpieces.

Dilations have exciting opportunities within CodeWorld that go beyond the standards. Advanced students can nurture their curiosity through the creation of famous fractals like the Sierpinski triangle. Furthermore, a student can use the slideshow feature in order to display the iterations of geometric transformation leading to the Sierpinski triangle. The student can calculate the area and perimeter of each iteration, and begin exploring the idea of a limit as the triangles’ perimeters going to infinity, and the area going to zero.
2.2.2. Embedded Project: Rotations

The supplemental lesson on rotations connects the ICT function `rotated()` to the 8th grade geometry standard 8.G.A.3\(^\text{18}\). Students explore the function in Unit 2, Lesson 2, immediately after the introduction of `dilated()`. In 8th grade, images are rotated about the origin of a Cartesian coordinate plane, just as they are in CodeWorld, as long as no other transformations have been applied.

In the current ICT curriculum, the supplemental lesson should be practiced after the students are introduced to the new function `rotated()` and before the students move onto the activities of the lesson.

Rotating an object about the origin is a difficult skill to visualize and learn for students in both Math and ICT classes. In the supplemental lesson, students begin with a LEAP 360 question and are then instructed to check their answer using the given model in CodeWorld. Students are instructed to modify the code in order to investigate the preimage rotating around the origin. After the fluency drill, the students should begin to visualize rotations as movements around the origin of the coordinate plane, similar to a rotation of a compass drawing tool rather than the rotation of a knob. Students will then investigate rotating objects that begin at the origin. These objects seem to be easier to rotate than those randomly placed in the plane since they follow our standard idea of rotating an object. Having students investigate rotation of differently placed images should help them with conceptual understanding of their assignments for the ICT course.

\(^{18}\) 8.G.A.3: Describe the effect of dilations, translations, rotations, and reflections on two-dimensional figures using coordinates. (Rotations are only about the origin, dilations only use the origin as the center of dilation, and reflections are only over the y-axis and x-axis in Grade 8.)
Rotations were one of the key features of LOGO that allowed students to push their creations to higher heights. The advanced student may research the history of LOGO and use CodeWorld in order to model the graphical outputs of their peers 30 years ago. At this point, the student will also investigate the difference between the procedural and functional computing paradigms. And finally, students can animate their rotations in order to create their own visual artifacts.

2.2.3. Embedded Project: Scalings

The supplemental lesson, Scalings, connects the ICT function `scaled()` to the 8th grade geometry standard 8.G.A.3\(^{19}\), with a focus on reflecting an object across the x-axis and/or y-axis. Students explore the `scaled()` function in Unit 2, Lesson 6, which is in the beginning of the first semester. Therefore, students are more likely to connect the concept in CodeWorld to their 8th grade knowledge.

In 8th grade, students learn to reflect a preimage across the x-axis and y-axis. This concept is demonstrated in CodeWorld by using the `scaled()` function, and inputting -1 for the horizontal scale parameter in order to reflect an image over the y-axis. Similarly, students can input -1 for the vertical scale in order to reflect an object over the x-axis.

In the supplemental lesson, students are to investigate the given model in order to create algebraic rules of an object’s vertices after undergoing scalings. Then, the students apply these algebraic ideas in order to answer LEAP 360-like questions. This

---

\(^{19}\) 8.G.A.3: Describe the effect of dilations, translations, rotations, and reflections on two-dimensional figures using coordinates. (Rotations are only about the origin, dilations only use the origin as the center of dilation, and reflections are only over the y-axis and x-axis in Grade 8.)
investigation should happen after the students are introduced to the \texttt{scaled()} function and before they begin the activities in the ICT lesson.

2.2.4. Embedded Project: Function Notation

The supplemental lesson about Function Notation connects the functions in CodeWorld to the Algebra I standard A1: F-IF.A\textsuperscript{20}. Students begin describing the structure of functions in Unit 3, Lesson 1, which is in the beginning of the second semester. At this point in the course, the students begin to enter more abstract computing concepts and so it is important to be able to make connections to the familiar function notation in Algebra I.

The embedded content should be practiced before the ICT lesson on Function Definitions is even opened.

The supplemental lesson begins by examining simple examples of code and having the students practice vocabulary regarding the functions’ mechanisms. Then, two figures are given, Figure A is a code that is typical for the skills practiced in the first semester. Figure B introduces the students to managing an algorithm’s complexity by creating a function. The following questions that the students are asked to answer are taken from the Algebra I Companion Document, a document the Louisiana Department of Education releases in order to help teachers prepare for the LEAP 2025 end of year assessment. And finally, students are given a glimpse of CodeWorld’s glossary. Although the syntax is unlike what is seen in a typical Algebra I course, the students are given the chance to relate the elements in the glossary to a function and its domain and

\textsuperscript{20} A1: F-IF.A: Understand the concept of a function and use function notation
range. This is an essential skill as a glossary can help bring to life ideas and graphics not directly addressed in class lessons.

2.3. Additional Algebra I Aligned Projects

The following lessons are labeled as additional projects because they can be used anywhere within the course after students have mastered the basic skills. These projects concentrate on enriching the students' math skills than programming skills. Each project contains a pre- and posttest and addresses multiple Algebra I standards.

2.3.1. Linear Versus Exponential Project

The Linear Versus Exponential Project uses CodeWorld in order to explore the following Algebra I state standards: F-IF.B.6, F-LE.A, and F-LF.B.5. A teacher may begin the project with a pre-test composed of questions from the LEAP 360 resources. The students will begin the project by comparing the rate of changes of a model created with CodeWorld's slideshow feature. Students extract data from the model, create a table to organize the data, and finally asked to plot the data in a spreadsheet. At this point, students have represented both a linear and an exponential function in four different ways and will practice problems that support the Algebra I curriculum further. Students will also have an opportunity to use the slideshow feature in CodeWorld, to model their own exponential or linear scenarios.

---

21 F-IF.B.6: Calculate and interpret the average rate of change of a linear and exponential function (presented symbolically or as a table) over a specified interval.
22 F-LE.A: Construct and compare linear, quadratic, and exponential models and solve problems
23 F-LF.B.5: Interpret expressions for functions in terms of the situation they model.
An advanced student could dive further into the idea of exponential functions by researching phenomena such as wildfires, population growth, or epidemics. Once a data set is found, a student can use exponential regression in order to find a line that best fits the data and then can use CodeWorld to model the scenario.

2.3.2. Systems of Equations Project

The Systems of Equations supplemental project begins with a model of two people and their savings accounts. The students will watch the bank accounts grow, then extract the data from the model and ultimately create a table, equations, and graphs in order to represent the simultaneous linear equations. This project connects to the Algebra I standard A1:A-REI.C²⁴

This project may be incorporated anywhere throughout the curriculum.

²⁴ A1:A-REI.C: Solve systems of equations
Chapter 3. Conclusions

In the summer of 2015, I was first introduced to CodeWorld in a mathematics graduate seminar course, that was co-taught by Mr. Fernando Alegre. In fact, other than a brief project using TRUBASIC in high school, this was my first exposure to computer programming. In the course we were exposed to a variety of isolated projects such as drawing a ball, then animating it to bounce, and a user-interactive project with a robot on a checkerboard. At the time, I had no real understanding what the objectives of this graduate course were, and I left the course excited to have coded but couldn’t quite articulate the importance of my newfound computer science knowledge to my daily practice as a mathematics teacher.

The following summer, in a follow-up graduate course, the number of participants at least doubled and the course now focused on learning the content of the LSU Introduction to Computational Thinking course in order to teach the content of the ICT course the following school year. This time around, we—the students—followed a set curriculum created by Mr. Alegre composed of projects within units and lessons. Benefiting from the refined nature of this course, I developed and retained the “how to's” of the programming environment CodeWorld, and engaged in questioning strategies with my peers, modeled by Mr. Alegre25.

LSU’s Introduction of Computational Thinking course is an actively evolving course as the course’s trained teachers and their students are continuously giving and receiving feedback. The current iteration of the course fully embraces the software

---

25 Such questions included, “how can you abstract the code further?” or “Keep going until you see the pattern”
development process and the development of computational thinking skills, in addition to the actual coding. These skills are practiced by having the students create a plan, then taking full ownership of the created content, and revising it as needed.

Just as the ICT course has evolved into the powerful course it is today, so will the connections evolve that we, as educators, are able to make across various high school curricula. A biology course could be enriched with additional ICT projects that support Louisiana biology model-based standards such as HS-LS1-4\textsuperscript{26} and HS-LS1-5\textsuperscript{27}. Also CodeWorld’s ample geometric capabilities can support high school geometry concepts with additional ICT projects that focus on clusters G.CO.A\textsuperscript{28} and G.CO.B\textsuperscript{29}.

Once students possess the fundamental CodeWorld skills, the environment can support courses outside of STEM education. For example, students can create scenes in CodeWorld to support the 9\textsuperscript{th} grade English standard “Make use of digital media in presentations to enhance understanding of findings, reasoning and evidence to add interest” (Louisiana Department of Education, 2019). In the ICT lesson, Data Manipulation, found in Unit 4, students practice skills to represent data sets with bar graphs and pie charts. These skills can be used to support a civics course. And there is the obvious art credit potential that students can gain through practice drawing and animating pictures.

The Introduction to Computational Thinking course is an engaging course where students practice and learn transferable foundational skills of programming through a

\begin{itemize}
\item \textsuperscript{26} HS-LS1-4: Use a model to illustrate the role of the cell cycle and differentiation in producing and maintaining complex organisms
\item \textsuperscript{27} HS-LS1-5: Use a model to illustrate how photosynthesis transforms light energy into stored chemical energy
\item \textsuperscript{28} G.CO.A: Experiment with transformations in the plane.
\item \textsuperscript{29} G.CO.B: Understand congruence in terms of rigid motions
\end{itemize}
text-based coding language. Because of the many graphical outputs one can create, students really have the opportunity to flourish in this programming environment. Since students already have a strong positive mindset towards the ICT course, they may not even recognize they are further investigating the standards from their math courses.

Through my time in LSU’s Masters of Natural Sciences program, I took math and computer science centered classes. The math courses I took at the graduate level reinforced many of the concepts introduced at the undergraduate level. The discussions and lectures opened my mind to questioning the concepts we teach in high schools while reigniting the intellectual curiosity I once had about mathematics and its role in the world around us. The math courses taken also inspired me to continue to practice linear algebra and calculus in my free time and find ways to integrate the concepts into high school lessons. The computing courses have played an integral role in my teaching career as I have taught, discussed, reflected upon, and envisioned the power of the Introduction to Computational Thinking course. I have also been inspired to continue to take Computer Science courses, and teach more computer science courses at the high school level. In gratitude to all the opportunity I have been given, I hope to always teach the ICT course along with aligned standards based lessons, wherever I may go, and inspire the ones around me to not just consume technology but create it.
Appendix A. ICT Course Outline

Introduction to Computational Thinking
Fall 2019/Spring 2020

Semester 1
Using code to represent objects and their relations
Unit 0: Introduction to Computing

1. Consent Forms
2. Surveys
3. Introduction to HTML
4. Pre-Test

Unit 1: The Software Development Cycle
Lists, Points, Shapes and Movement

1. The CodeWorld IDE
2. Pictures and Anchors
3. Translations
4. Overlays and Lists
5. Basic shapes and colors
6. Design Techniques

Project: Illustration

Supplemental lessons:
1. Other polygonal functions
2. Fake 3D drawings

Unit 2: Abstraction and Decomposition
Functions, Operators, Trees and Grids

1. Function Applications and Trees
2. Dilations and Rotations
3. Operators
4. The Overlay Operator
5. Scope, Local Definitions and Data Dependency
6. Scalings
7. Unevaluated expressions and Grids

Project: Quilt Patterns

Midterm Project: Diagrams
Semester 2
Applying Computational Thinking to Solve Problems
Unit 3: Patterns and Regularity

1. Function Definitions
2. How to Design Functions
3. Points and Regular Polygons
4. Random Numbers
5. Drawing a Clock
6. Combining Transformations
7. Graphs and Irregular Grids

Project: A Neighborhood Supplemental

lessons:
1. Equilateral triangles and honeycombs
2. Stadiums and Squircles

Unit 4: Data and Calculations

1. Slideshows and Sliders
2. Areas of Rectangles and Circles
3. List and Tuple manipulation
4. Charts
5. Text processing
6. Integer numbers
7. Dollar amounts

Project: Interpreting Data

Unit 5: Models in Space and Time

1. Computational Thinking concepts
2. Introduction to animations
3. More about Animations

Project: Animation Clip

Final Project: Movable Parts

The full curriculum can be viewed at https://blackeyepeas.phys.lsu.edu/Review/RPP-2019/ICT/.
Appendix B. Embedded Projects

Dilations

**Louisiana State Standard: 8.G.A.3**: Describe the effect of dilations, translations, rotations, and reflections on two-dimensional figures using coordinates.

**Activity 1:**
Use the following code to explore the vertices of dilated triangles:
https://blackeyepeas.phys.lsu.edu/cwn/#PK2jCJgH1qF_wHDpIUMHkJA

1. List the coordinates of the vertices of the preimage_1 and image_1. How do the coordinates of the two figures relate to the dilation factor?

2. List the coordinates of the vertices of the preimage_2 and image_2. How do the coordinates of the two figures relate to the dilation factor?

Use the relationship between coordinates and dilation factor to help you determine the next question:

3. Suppose triangle PQR has coordinates (-2, -4), (0,5), and (3,7). The triangle is then dilated with a center at the origin and by a factor of 3 to create triangle P′Q′R′. What are the coordinates of P′, Q′, and R′?

4. Create a preimage_3 and image_3 in order to model out your math. Then, paste a link to your code here:

**Activity 2:**
Use the following link to explore side lengths of dilated triangles:
https://blackeyepeas.phys.lsu.edu/cwn/#P4v0aJZeD_6Flboa0m_YnJQ

1. Begin by reading through and making sense of the code. Which lines will be used to change the vertices of the right triangle?

2. Use these lines and input any real number between 0 and 5. Now record the length of each leg and the hypotenuse of the pre-image.
3. Using your knowledge and skills from 8th grade, apply the Pythagorean Theorem and verify the program’s approximation of the hypotenuse.

4. Now, display the image and record the image’s leg lengths and hypotenuse.

5. How do the side lengths of the preimage and image relate to the dilation factor?

Activity 3:
Use the following link to explore angle measurements of dilated triangles: https://blackeyepeas.phys.lsu.edu/cwn/#PTkd_jVP0JkG6OMRBT2LWw

1. Modify the code in order to create your favorite right triangle as your preimage.
   -- Angle A is the top vertex, angle B is the right vertex, and angle C is the right angle --
   List the measurement of each angle:

2. Now display the image and its measurements and record the measurement of each angle.

3. Change the dilation factor in order to shrink the preimage. Do the angle measurements follow the same rule?

4. Paste your final workspaces code here:

5. Write a conclusion regarding angle measurements and dilations.

Activity 4:
Apply the concepts!
1. Suppose triangle ABC where side AB=12. After a dilation is applied, A’B’=4. What was the dilation factor used?
2. True or false: A 5-sided star is magnified by a factor of 10. Therefore, the stars side lengths and angle measurements both increased by x10.

3. Suppose a square PQRS has point P at the origin and point R at (4,4). After being dilated by a factor of 3, what is the image's perimeter?

Activity 5:

1. Suppose your preimage is a circle with radius 1, and center point at (2,2). Predict the image after the circle is dilated from the origin by a factor of 3. Be sure to predict both the center point and radius.

2. Create a model in CodeWorld that supports Activity 5, Question 1. Then, paste your code here:
Rotations

**Louisiana State Standard: 8.G.A.3:** Describe the effect of dilations, translations, rotations, and reflections on two-dimensional figures using coordinates.

**In order to understand how rotations in CodeWorld work, we should recall rotation skills learned in 8th grade:**

**Activity 1:**
1. Rectangle PQRS is graphed on the below coordinate plane:

![Coordinate Plane with Rectangle PQRS](image)

Rectangle PQRS is rotated 90° counterclockwise about the origin to create rectangle P'Q'R'S (not shown). What are the coordinates of R'?

Click your prediction here:
- a. (-7,6)
- b. (7,6)
- c. (-6,7)
- d. (6,7)

2. Use the below code to help you verify your answer:
   [https://blackeyepeas.phys.lsu.edu/cwn/#PXnJXDD1HyrpwVy7G-Nx3ZQ](https://blackeyepeas.phys.lsu.edu/cwn/#PXnJXDD1HyrpwVy7G-Nx3ZQ)

3. Create new identifiers in order to create at least 6 more copies of the rotated preimage. Display your preimage and all the new images at once. Choose rotation values between 90 and 360 in order to fully investigate the rotation around the origin. Then, paste your code here:
Activity 2:
When the preimage is centered at the origin, rotations look like they behave differently. In some instances, it may be easier to control:

1. Create your own wobbly letters name by rotating your letter from the center. Then translate each letter to the desired position. Paste a link to your code here:

Example code:
https://blackeyepeas.phys.lsu.edu/cwn/#PPJTODoYwmjqCaAiR8UnaOQ
Scalings

**Louisiana State Standard: 8.G.A.3:** Describe the effect of dilations, translations, rotations, and reflections on two-dimensional figures using coordinates.

In order to understand how reflections in CodeWorld work, we should recall reflection skills learned in 8th grade:

**Activity 1:**

1. Rectangle PQRS is graphed on the below coordinate plane:

   ![Coordinate Plane](image)

   Rectangle PQRS is reflected across the y-axis in order to create rectangle P'Q'R'S(not shown). What are the coordinates of P'?
   
   Click your prediction here:
   a. (-6,0)
   b. (6,0)
   c. (-6, -2)
   d. (-6,2)

2. Use the below code to help you verify your answer:
   [https://blackeyepeas.phys.lsu.edu/cwn/#PIpqgvf5jb5H_7PWdFxDkBQ](https://blackeyepeas.phys.lsu.edu/cwn/#PIpqgvf5jb5H_7PWdFxDkBQ)

3. Describe the relationship between the coordinates of the preimage and the image and how their relationship to the arguments of the scaled function. Simply stated, how does the scaled function work?

4. Experiment with the above code in order to determine how to reflect an object over the x-axis. Paste a link to the image P'Q'R'S' reflected over the x-axis into quadrant 1:

5. Continue to experiment with the scaled function, by replacing the horizontal_scale value with 0. Why does the image disappear?

6. Suppose triangle CAT has coordinates C(5,4), A(1,4) and T(1,8). The triangle now undergoes a horizontal scale factor of -2, and a vertical scale factor of What is the coordinate of C'
Activity 1:

Observe and execute the following code:

1. Describe the program’s output:

   Each computing language has its own predefined functions, identified by reserved keywords. In CodeWorld, these built-in functions light up **bold** and **blue**. Each function will take in a value, ultimately resulting in an output.

2. Explain the code in English from Question 1:

   “When _____ is inputted into the function “ _____ ”, a _________________ is outputted”

Activity 2:

Often times a programmer will want to encapsulate a task, meaning condense multiple instructions into a single line of code. In this case, a programmer can and should write their own function.

For example: Suppose in CodeWorld we want to create a scene with many circles of various sizes and colors. Rather than continuously repeating the “colored” and “circle” functions (Figure 1), one could write their own function, followed by **parameters**, or input values of the function (Figure 2).

**The programmer may name the function and parameters anything that follows the identifier rules of naming functions and variables, but keep in mind the names given should be meaningful**
1. Describe in words how lines 4-6 are working from figure 2.

2. Create a function that manages the complexity of an algorithm that generates multiple rectangles of different sizes, with different translations across the canvas. How many parameters will your function take in?

3. Use your function display at least 5 rectangles. Paste a link to your code here:

4. Change your function in order to generate squares instead of rectangles. How many parameters will this new function take in? Paste a link to your code here:

**Activity 3:**
In algebra, we define a function as the correspondence between two sets, $X$ and $Y$, in which each element of $X$ is assigned to one and only one element of $Y$. The set $X$ is called the domain of the function, while $Y$ is called the range. Elements of the range are generated after an input undergoes a function’s rule.

Unlike CodeWorld, functions in algebra are named with one letter, and an often ambiguous parameter, most commonly seen as $f(x)$. Despite this notation, functions in algebra class behave just like functions in CodeWorld.

For example, let $f(x) = 3x + 7$. If asked to evaluate $f(2)$, one would simply pass the input of 2 through the function rule of $3x+7$. Simply put, $f(2)=3(2)+7$, thus $f(2)=13$.

1. Let $g(x) = x^2 + 3$
   a. Evaluate $g(5)$.
   b. Evaluate $g(0)$.
   c. Evaluate $g(input)$
   d. Evaluate $g(p+1)$. 
2. Let \( h(x, y, z) = x + y^2 + x^3 \). Evaluate \( h(3, 4, 1) \).

3. You placed a yam in the oven and, after 45 minutes, you take it out. Let \( f \) be the function that assigns to each minute after you placed the yam in the oven, its temperature in degrees Fahrenheit.

Write a sentence for each of the following to explain what it means in everyday language.

a. \( f(0) = 65 \)

b. \( f(5) < f(10) \)

c. \( f(40) = f(45) \)

d. \( f(45) > f(60) \)

Activity 4:
In algebra, the notation \( f: X \rightarrow Y \) is used to name the function and describes both \( X \) and \( Y \). If \( x \) is an element in the domain \( X \) of a function \( f: X \rightarrow Y \), then \( x \) is matched to an element of \( Y \) called \( f(x) \).

This notation is similar to the glossary of CodeWorld:

- `circle :: Number -> Picture`
- `solidCircle :: Number -> Picture`
- `thickCircle :: (Number, Number) -> Picture`
- `rectangle :: (Number, Number) -> Picture`
- `solidRectangle :: (Number, Number) -> Picture`
- `thickRectangle :: (Number, Number, Number) -> Picture`
- `lettering :: Text -> Picture`

1. A curious student is looking up in the reference glossary how to use not predefined colors and sees \( \text{RGB} :: (\text{Number}, \text{Number}, \text{Number}) \rightarrow \text{Color} \). Predict how this function will work. Then, color a `solidCircle` mint green. Paste a link to your code here:
Appendix C. Algebra I Aligned Projects

Linear VS Exponential Pre and Post Test

1. **Part A** Observe the below table and create an equation to represent each set of data using N to represent the number of months:

<table>
<thead>
<tr>
<th></th>
<th>Initial Value</th>
<th>1 month</th>
<th>2 months</th>
<th>3 months</th>
<th>4 months</th>
<th>5 months</th>
<th>N months</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of ladybugs</td>
<td>50</td>
<td>100</td>
<td>150</td>
<td>200</td>
<td>250</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>Number of bumblebees</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td></td>
</tr>
</tbody>
</table>

Number of ladybugs equations:

Number of bumblebees equation:

**Part B:**
At what month will the number of bumblebees be greater than the number of ladybugs?

**Part C:**
Use your equations to predict how many of each bug will be present in 2 years

2. Carol shares her joke with 3 of her friends. The next day, each of her friends tells the joke to 4 other people. The next day, each person that heard the joke the previous day tells the joke to four different people. The expression $3(4)^x$ models the situation where $x$ represents the number of days after Carol initially told the joke. What do the different parts of the expression of the expression represent?

Complete each sentence by choosing the correct phrase to write on each line.

3 represents ____________________________________________.
4 represents ____________________________________________.
3. Jose bought a new car 2 years ago for $18,000. The value of his car decreased exponentially and is now valued at about $12,400.

Part A
Which function estimates the value of Jose’s car after x years since buying the car?
A. \( f(x)=12,400(0.17x) \)
B. \( f(x)=12,400(0.83x) \)
C. \( f(x)=18,000(0.17x) \)
D. \( f(x)=18,000(0.83x) \)

Part B
Jose predicts that the value of his car will decrease exponentially by 25% over the next 4 years. If Jose’s prediction is correct, about how much will his car be valued at, to the nearest thousand dollars, after 4 more years?

4. The table shows data for a linear and an exponential function.

<table>
<thead>
<tr>
<th>x</th>
<th>f(x)</th>
<th>g(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>9</td>
<td>385</td>
</tr>
<tr>
<td>3</td>
<td>27</td>
<td>585</td>
</tr>
<tr>
<td>4</td>
<td>81</td>
<td>785</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Explain how you know which function models a linear function and which function models an exponential function. In your explanations, describe how to determine \( f(x) \) and \( g(x) \) when \( x=5 \).
5. Amy has invested money in an account. The graph depicts her balance in the account given the number of years since she first invested in an account. The graph depicts her balance in the account given the number of years since she first invested money in the account.

[Graph: Balance in Amy’s Investment over Time]

What is the approximate average rate of change in the balance of the account from Amy’s initial investment through year 6?

A. $10 per year
B. $16 per year
C. $42 per year
D. $100 per year

6. At the beginning of an experiment, the number of bacteria in a colony was counted at time t=0. The number of bacteria in the colony t minutes after the initial count is modeled by the function \( b(t) = 4(2)^t \). What is the average rate of change in the number of bacteria for the first 5 minutes of the experiment?

Select from the list to correctly complete the sentence:

The average rate of change in the number of bacteria for the first 5 minutes of the experiment is ____________

<table>
<thead>
<tr>
<th>Blank 1 Choices</th>
<th>Blank 2 Choices</th>
</tr>
</thead>
<tbody>
<tr>
<td>24.0</td>
<td>bacteria</td>
</tr>
<tr>
<td>24.8</td>
<td>minutes</td>
</tr>
<tr>
<td>25.4</td>
<td>Bacteria per minute</td>
</tr>
<tr>
<td>25.6</td>
<td>Minutes per bacteria</td>
</tr>
</tbody>
</table>
7.

Elephant Population Estimates- Namibia
Combined estimates for Etosha National Park and the Northwestern Population

The elephant population in Northwestern Namibia and Etosha National Park can be predicted by the expression \(2,649(1.045)^x\) where \(x\) is the number of years since 1995.

What does the value of 2,649 represent?

A. The predicted increase in the number of elephants in the region each year
B. The predicted number of elephants in the region in 1995
C. The year when the elephant population is predicted to stop increasing
D. The percentage the elephant population is predicted to increase each year
Linear Versus Exponential Project

Objectives:
SWBAT interpret expressions for functions in terms of the situation they model.
SWBAT make predictions based on linear and exponential models
SWBAT determine the average rate of change in terms of the context of the data given

Activity 1:
Use the below code to answer the following questions:
https://blackeyepeas.phys.lsu.edu/cw/#PMcMDDgPW09SMvTLEKTLI-q

Q1: Click through the output panel and watch the number of dots grow. Describe the rate of change for each set of dots.

Q2: At which slide number do the green dots outnumber the blue dots?

Q3: Complete the below table, including the future numbers of slides:

<table>
<thead>
<tr>
<th></th>
<th>Initial Value</th>
<th>Slide 1</th>
<th>Slide 2</th>
<th>Slide 3</th>
<th>Slide 4</th>
<th>Slide 5</th>
<th>Slide 6</th>
<th>Slide 7</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Blue Dots</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of Green Dots</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q4: Describe how you predicted how many dots would be on slide 7:

Q5: The blue dot's growth represents a linear relationship, which can be represented by the equation \( y=ax+b \), where \( a \) is the rate of change and \( b \) is the initial value. Write an equation to represent the number of blue dots with \( x \) representing the slide number and \( y \) is the total number of blue dots on the screen.

Q6: The green dot’s growth represents an exponential relationship which can be represented by the equation \( y=ab^x \) where \( a \) is the initial value and \( b \) is the rate of change. Write an
equation to represent the number of green dots with x representing the slide number and y is the total number of blue dots on the screen.

Q7: Test out your equations by determining if they work for the values in the table. Then, use your equations to predict how many dots would appear on slide 24, assuming the output panel was infinitely tall and wide.

Q8: Plot your data in a spreadsheet in order to graph each curve digitally on the same plane. Be sure to label the axes and color each graph accordingly. Then, paste your graphs here.

Activity 2:

Part A:
Read the following scenarios and classify them as linear or exponential. Then, write an equation to model the scenario:

Scenario 1: A music artist makes $400 per month from their recording studio, and makes an additional $7 per album sold.

Scenario 2: Bacteria cells are known to rapidly increase their population size. Under controlled and favorable conditions, 1 bacteria cell will double at regular intervals.

Scenario 3: In medicine, a half-life is defined as the period of time required for the concentration or amount of drug in the body to be reduced by one-half. Suppose you have drunk a standard cup of coffee which contains roughly 100 mg of caffeine. Determine the amount of caffeine in your system after a certain number of half-life intervals.

Scenario 4: You are draining 20 gallons of water from your bathtub and the tub drains at 2 gallons a minute.
Part B:
Observe the following graphs. Create your own scenario each graph may represent.

i. Possible scenario (Be sure to include an initial value and rate of change):

ii. Possible scenario (Be sure to include an initial value and growth rate):

Part C: Create a slideshow in order to model each scenario from Part B. Paste links to your code here:

Activity 4: Use the below code to answer the following question:
At what slide number will the squares fill the screen? (HINT: Avoid brute force by setting up an equation)

Activity 5:
Create a slideshow (at least 8 slides) of images which model a linear or exponential growth scenario. Write your scenario in the beginning, be sure to use -- to comment it out. Below your commented out scenario, write the equation of your scenario. Your growth rate may NOT be +2 or *2. You may use the following codes to help you get started:

Linear: https://blackeyepeas.phys.lsu.edu/cw/#P2EEEProomBxTFHA2uNYMFQ

Exponential: https://blackeyepeas.phys.lsu.edu/cw/#PpuznpclflJoekdXrzR5wA

Bonus!
Include closed captioning on your project
Pre and Post Test for Systems of Equations

1. What is the x-coordinate in the solution of the system of equations?
\[
\begin{align*}
\frac{2}{3}x = y \\
4x - 9y = -20
\end{align*}
\]

2. What is the solution to the system of equations?
\[
\begin{align*}
-5 &= x - 3y \\
11 &= -3x + \text{ }\text{ }
\end{align*}
\]

Write the appropriate numbers to create the solution:
\[
(_______, _______)
\]

3. Consider the system of equations:
\[
\begin{align*}
-2x + 6y &= -8 \\
x + 3y &= -4
\end{align*}
\]

What value of c would produce a system that has an infinite number of solutions? Justify your answer.

4. The perimeter of Dana’s rectangular garden is 61 feet. The length of her garden is 5 feet more than twice the width:
   - Create a system of equations that can be used to determine the length and width of the garden.
   - Be sure to define your variables
   - What is the length and width of the garden? Show all your work.
5. The equation $y_1 = x^2 - 4x$ represents Function 1. The graph of Function 1 is shown:

Function 2 is represented by the equation $y_2 = x - 6$. Which values are solutions to $y_1 = y_2$?

Select each correct solution:

A. $x = 6$
B. $x = 3$
C. $x = 2$
D. $x = 1$
E. $x = 0$
F. $x = -6$

6. Malik earns $9.50 per hour at his job. He deposits everything he earns into his savings account. He currently has $100 in his savings account. Linda earns $7.00 per hour at her job. She deposits everything she earns into her savings account. She currently has $320 in her savings account.

- Write a system of equations that can be used to determine when Malik and Linda will have an equal amount of money in their accounts after working the same number of hours.
- How many work hours will it take for Malik and Linda to have an equal amount of money in their accounts? Show your work.
7. The graphs of \( f(x) = -x^3 - 4 \) and \( g(x) = 0.5x^2 + x + 4 \) are given.

Use the graphs to find the solution to the equation \(-x^3 - 4 = 0.5x^2 + x + 4\):

Solution: \( x = \)
Systems of Equations Project

Objectives:
SWBAT model a system of equations using a table, graph, equation, and computer model
SWBAT solve a system of equations
SWBAT use technology to find the solution to a system of equations

Malik earns $12.00 per hour at his job. He deposits everything he earns into his savings account. He begins with $84 in his savings account. Lynn earns $8.00 per hour at her job and also deposits everything she earns into her savings account. She begins with $136.00 in her savings account.

Use the below code to answer the following questions:
https://blackeyepeas.phys.lsu.edu/cw/#P-hTd26rOwqz5N89DZHLQTO

Q1: Click through the output panel and watch the bank accounts grow. Identify each person’s initial value and describe the rate of for each person’s bank account.

Q2: Create a table that extracts the data from the slides. Keep in mind, the output panel doesn’t display slide 0:

<table>
<thead>
<tr>
<th>Slide number (n)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>M(n)</td>
<td>$84</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L(n)</td>
<td>$136</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q3: At which week are the quantities equal? What is the value at this week? Write your answer in a complete sentence.

Q4: The answers from Q3 is the solution to the system of linear equations that model Malik and Lynn’s bank accounts. Write your answers an ordered pair.
Q5: Write a conclusion about how to determine the solution to a system of linear equations in a table:

Q6: Input your data into a spreadsheet. Then create a graph using the data sets. Be sure to label your axis and include a key. Paste your final product here:

Q7: Write a conclusion regarding finding a solution to a system of equations on a graph.

Q8: Write a function to represent each of the people’s bank accounts. Use n to represent the slide number/week number. Hint! Remember linear equations are in the form ax+b, where a is the rate of change and b is your initial value.

Q9: Think about it! We wanted to find the point in which the equations were equal. Set your functions from Q9 equal to one another and solve for n. Once you arrive at a solution, plug the value for n into each function and verify your outcome.

Q10: Sometimes when solving a system of equations there may be infinite solutions or no solutions. Think back to our original question, we were trying to determine when Malik and Lynn would have the same amount of money in their account.

**Part A** Describe a scenario where Malik and Lynn will never have the same amount of money! (Think initial value and rate of change)

**Part B** Describe a scenario where Malik and Lynn will always same amount of money!

**CHALLENGE** Modify the code in order to represent each of your scenarios.
Systems of Equations Exit ticket:

1. The graphs of \( f(x) = -x^2 - 4 \) and \( g(x) = 0.5x^2 + x + 4 \) are given:

   ![Graphs of f(x) and g(x)]

   Use the graphs to find the solution to the equation \(-x^2 - 4 = 0.5x^2 + x + 4\)

2. Consider the following system of equations:
   \[
   y = 4x - 12 \quad \text{and} \quad y = 8x + 28
   \]

   Find the \( x \)-coordinate of their solution:

3. Ryan earns $9.50 per hour at his job. He deposits everything he earns into his savings account. He currently has $100 in his savings account. Linda earns $7.00 per hour at her job. She deposits everything she earns into her savings account. She currently has $320 in her savings:
   - Write a system of equations that can be used to determine when Ryan and Linda will have an equal amount of money in their accounts after working the same number of hours.
   - How many work hours will it take for Ryan and Linda to have an equal amount of money in their accounts?
References


Louisiana Department of Education. (2019) Student Standards for English Language Arts.


Piaget, J. (1973). To understand is to invent: The future of education. Pg. 106


Vita

Michelle Laskowski was born in Hinsdale, Illinois, daughter of Robert and Brigitte Laskowski. She graduated from York Community High School in May 2011. Following graduation, she attended DePaul University in Chicago for two academic years, before transferring to Louisiana State University. In December 2015, she earned a Bachelor of Science degree in Mathematics with a concentration in secondary education. She entered the Louisiana State University Graduate School in June 2017 and is a candidate for the Master of Natural Sciences degree. She currently teaches Algebra I and Advanced Placement Computer Science Principles at Central High School in Baton Rouge, Louisiana. She plans to continue her education of computer science and participate in the LSU Computing Pathway.