

CODING IN THE CURRICULUM: LEARNING COMPUTATIONAL  
PRACTICES AND CONCEPTS, CREATIVE PROBLEM  
SOLVING SKILLS, AND ACADEMIC CONTENT IN  
TEN TO FOURTEEN-YEAR-OLD CHILDREN

---

A Dissertation  
Submitted to  
the Temple University Graduate Board

---

In Partial Fulfillment  
of the Requirements for the Degree  
DOCTOR OF PHILOSOPHY

---

by  
Kevin S. Donley  
Diploma Date August 2018

Examining Committee Members:

Frank Farley, Advisory Chair, Psychological Studies in Education  
Joseph DuCette, Office of the Dean, Psychological Studies in Education  
Yasmin Kafai, Graduate School of Education, University of Pennsylvania  
John Hall, External Member, Policy, Organizational, & Leadership Studies

ProQuest Number:10842428

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 10842428

Published by ProQuest LLC (2018). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code  
Microform Edition © ProQuest LLC.

ProQuest LLC.  
789 East Eisenhower Parkway  
P.O. Box 1346  
Ann Arbor, MI 48106 – 1346

©  
Copyright  
2018

by

Kevin Donley  
All Rights Reserved

## ABSTRACT

The fundamentals of computer science are increasingly important to consider as critical educational and occupational competencies, as evidenced by the rapid growth of computing capabilities and the proliferation of the Internet in the 21<sup>st</sup> century, combined with reimagined national education standards. Despite this technological and social transformation, the general education environment has yet to embrace widespread incorporation of computational concepts within traditional curricular content and instruction. Researchers have posited that exercises in computational thinking can result in gains in other academic areas (Baytak & Land, 2011; Olive, 1991), but their studies aimed at identifying any measurable educational benefits of teaching computational concepts to school age children have often lacked both sufficient experimental control and inclusion of psychometrically sound measures of cognitive abilities and academic achievement (Calao, Moreno-León, Correa, & Robles, 2015). The current study attempted to shed new light on the question of whether using a graphically-based computer coding environment and semi-structured curriculum –the Creative Computing Course in the Scratch programming language –can lead to demonstrable and significant changes in problem solving, creative thinking, and knowledge of computer programming concepts.

The study introduced 24 youth in a summer educational program in Philadelphia, PA to the Scratch programming environment through structured lessons and open-ended projects for approximately 25 hours over the course of two weeks. A delayed treatment, control trial design was utilized to measure problem solving ability with a modified version of the Woodcock-Johnson Tests of Cognitive Abilities, Fourth Edition (WJ-IV),

Concept Formation subtest, and the Kaufman Tests of Educational Achievement, Third Edition (KTEA-3) Math Concepts and Applications subtest. Creative problem solving was measured using a consensual assessment technique (Amabile, 1982). A pre-test and post-test of programming conceptual knowledge was used to understand how participants' computational thinking skills influenced their learning. In addition, two questionnaires measuring computer use and the Type-T (Thrill) personality characteristic were given to participants to examine the relationship between risk-taking or differences in children's usage of computing devices and their problem solving ability and creative thinking skills.

There were no differences found among experimental and control groups on problem solving or creative thinking, although a substantial number of factors limited and qualified interpretation of the results. There was also no relationship between performance on a pre-test of computational thinking, and a post-test measuring specific computational thinking skills and curricular content. There were, however, significant, moderate to strong correlations among academic achievement as measured by state standardized test scores, the KTEA-3 Math Concepts and Applications subtest, and both the pre and post Creative Problem Solving test developed for the study. Also, higher levels of the Type T, or thrill-seeking, personality characteristic were associated with lower behavioral reinforcement token computer "chips," but there were no significant relationships among computer use and performance on assessments.

The results of the current study supported retention of the null hypothesis, but were limited by small sample size, environmental and motivational issues, and problems with the instrumentation of the curriculum and selected measures. The results should,

therefore, not be taken as conclusive evidence to support the notion that computer programming activities have no impact in other areas of cognitive functioning, mathematic conceptual knowledge, or creative thinking. Instead, the results may help future researchers to further refine their techniques to both deliver effective instruction in the Scratch programming environment, and also target assessments to more accurately measure learning.

This work could not have been done without the instrumental and unconditional support from my mother, Deb Donley, who has not only supported me throughout the long and often stressful journey through graduate school, but also has remained an ever-positive, guiding force throughout my life. Thank you does not even begin to touch upon the level of gratitude I owe to you.

## ACKNOWLEDGMENTS

This work was an enormous undertaking that often involved collaboration with a variety of researchers and educators at various stages along the way. Without the support and guidance of those listed below, this work could not have been possible. Thank you to Christopher Hanlon, Shanisha Mitchell, and Avery Thornburg of Belmont Charter Network for hosting the study and providing logistical support; Eljakim Schrijvers and Daphne Blokhuis for helping coordinate the Bebras Challenge Computational Thinking Assessment; the PhilaSoup organization for providing funds to purchase individual Creative Computing Curriculum workbooks for students; Kelsey Huse for going above and beyond as lead course instructor; the teams of research assistants who acted as field assessors and raters of student responses; Catherine (Cathy) Fiorello for help with modifying the WJ-IV CF; James Kaufman for providing direction on the assessment of creativity; Wilhelmina (Willa) Peragine for consulting on the implementation of Creative Computing Course; Jill Denner, Shannon Campe, and Deborah (Debby) Fields for steering me in the right direction in searching the literature; the Computer Recycling Center at Temple University for letting me repurpose their computer chips; and my advisory and examining committee members for providing guidance, feedback, and support throughout the study. Last but not least, I thank my wife, Caitlin Martin, from the bottom of my heart for always being there to troubleshoot a problem, listen to me vent, and cheer me on through the countless nights spent sequestered in front of the computer.



## TABLE OF CONTENTS

	Page
ABSTRACT .....	iii
DEDICATION .....	vi
ACKNOWLEDGMENTS .....	vii
LIST OF TABLES .....	xi
LIST OF FIGURES .....	xii
CHAPTER	
1. INTRODUCTION .....	1
2. REVIEW OF THE LITERATURE .....	5
Historical Developments in Educational Technology .....	13
Computational Thinking as a 21 <sup>st</sup> Century Literacy .....	23
How Can Educators Best Support the Development of CT Skills? .....	26
Games and Game-based Learning .....	29
Peer Critique and Collaboration .....	31
What Can Students Learn While Participating in Programming Activities? .....	33
Programming Conceptual Knowledge .....	33
Problem Solving Skills .....	37
Metacognitive Skills .....	39
Academic Skills and Content .....	40
Creative Thinking Skills .....	44
Assessing Learning in Computational Thinking .....	47
State of the Field .....	53

3. METHODS .....	55
Participants.....	55
Design .....	59
Measures .....	62
Materials .....	71
Data Collection .....	75
Relationship Among Variables.....	84
4. RESULTS .....	90
RQ1: Problem Solving and Creative Thinking.....	90
RQ2: Computational Thinking Skills .....	92
5. DISCUSSION.....	97
Notable Findings.....	97
No Group Differences Across Assessment Variables .....	97
Difficulty of Teaching the Creative Computing Curriculum in an Informal Learning Environment .....	100
Difficulty Assessing Computational Thinking .....	104
Limitations .....	107
Conclusion .....	112
REFERENCES CITED.....	115
APPENDICES	
A. CREATIVE PROBLEM SOLVING PRE-TEST .....	135
B. CREATIVE PROBLEMS SOLVING POST-TEST.....	139
C. TYPE T PERSONALITY QUESTIONNAIRE.....	143

D. USA BEBRAS 2016 QUESTIONS AND ANSWERS .....	144
E. PROGRAMMING CONCEPTUAL KNOWLEDGE POST-TEST .....	164
F. CORRELATIONS BY OVERALL SAMPLE .....	171
G. CORRELATIONS BY CONTROL GROUP .....	172
H. CORRELATIONS BY EXPERIMENTAL GROUP .....	173

## LIST OF TABLES

Table	Page
3.1. 2016 USA Bebras Challenge (PCK <sub>1</sub> ) Assessment Composition.....	68
3.2. Programming Conceptual Knowledge Post-test (PCK <sub>2</sub> ) Composition.....	70
3.3. Day-by-day Lessons for Experimental and Control Groups.....	73
3.4. Creative Problem Solving Pre-test (CPS <sub>1</sub> ) Interrater Reliability Statistics .....	78
3.5. Creative Problem Solving Post-test (CPS <sub>2</sub> ) Interrater Reliability Statistics .....	78
3.6. Perceived Misinterpretations Across Items on CPS <sub>1</sub> and CPS <sub>2</sub> .....	79
3.7. Youth Computer Activities at Home .....	81
3.8. Youth Computer Activities at School .....	82
3.9. Descriptive Statistics Across Assessments by Group .....	84
3.10. Correlations among PSSA, KTEA-3 MCA, and CPS scores .....	87
4.1. Mann-Whitney U Test Statistics Across Assessments and Group .....	91
4.8. Mann-Whitney U Test Statistics for Change in Assessment Scores .....	91

## LIST OF FIGURES

Figure	Page
3.1. Experimental Design and Measures.....	62
4.1. Mean ratings on the CPS assessments .....	92
4.2. Ranked Order of PCK <sub>1</sub> Scores by PCK <sub>2</sub> Scores Across Students .....	93
4.3. Number of Correct Responses Across PCK <sub>1</sub> Items .....	94
4.4. Number of Correct Responses Across PCK <sub>2</sub> Items .....	94

## CHAPTER 1

### INTRODUCTION

The digital age requires competence in a variety of skills in order to be successful, some directly related to academics and others indirectly related to academics through creative thinking and problem solving. One of the most highly valued skills in the age of information is the ability to quickly and efficiently solve problems (Federation of American Scientists, 2006); therefore, students entering the workforce should be prepared to meet this need. Providing the appropriate educational supports for the development of problem solving abilities is not only important for an individual's success in many aspects of life, but also for the collective future of the nation and world at large (National Research Council, 2010). The field of computer science may be one avenue in which students can learn problem solving skills as it is inherently intertwined with solving complex problems using language and logic (diSessa, 2000). Computer programming even in the most rudimentary sense has the potential to be an influential, creative, and engaging process that can lead to deeper understanding of the physical world (Papert, 1980). Furthermore, computer-programming activities involve an iterative problem solving approach that mimics the scientific method itself (Tracz, 1979).

According to the US Department of Labor and Statistics (2013) computer systems design and related service occupations are predicted to add over 600,000 jobs by 2022, which represents six percent of expected job growth across industries. Additionally, occupations in which an understanding of central principles of computer science is listed as a requisite skill are evident across industries, with 67% of these jobs positioned outside of the tech sector (Carnevale, Smith, & Melton, 2011). Clearly, there is demand to

educate individuals in the fundamentals of computer science not only to meet the needs of the occupational landscape of the modern era, but also to foster the development of critical problem solving skills that can be applied to a variety of situations.

Although there is a need to train technically skilled individuals to solve problems with computers, and a potential benefit by increasing problem solving abilities, general education in the United States does not put the same kind of emphasis on computer science as it does for traditional academic subjects (National Science Foundation, 2009). Of the nearly 42,000 high schools in the United States, just over 3,200 offered AP courses in Computer Science in the 2015-2016 school year (College Board, 2015). A recent large-scale, multiyear research effort conducted by Gallup including students ranging from 7<sup>th</sup> to 12<sup>th</sup> grade, parents, K-12 teachers, and school administrators (principals and superintendents) found that while both parents and students viewed computer science as just as important as other classes, e.g., history, math, etc., principals and superintendents did not perceive a high demand from students or parents for computer science courses in their districts; moreover, less than one third of teachers, principals, and superintendents surveyed reported that computer science education is currently a top priority for their school or district (Gallup, 2014).

Some claim that students' problem solving abilities can directly benefit from teaching computer science in an engaging way (Ackaoglu, 2014; Ackaoglu & Koehler, 2014; Au & Leung, 1991; CSTA, 2011; De Corte, 1992; diSessa, 2001; Hwang, Hung, & Chen, 2013; Ioannidou, Repenning, & Webb, 2009; Khasawneh, 2009; Li, 2010). Others contend that there are also indirect benefits to more traditional academic subjects through more highly developed metacognitive abilities (Allsop, 2015; Clements & Nastasi, 1999;

Li, 2010), and creative thinking skills (Clements, 1986; Kim, Chung, & Yu, 2013). Recognition that understanding fundamental principles of computer science is of great importance in education has come to light with the National Research Council's (NRC) publication of *A Framework for K-12 Science Education*; specifically, the NRC's decision to list computational problem solving as one of the eight essential practices for the scientific and engineering dimension (NRC, 2012). In response, non-profit organizations and professional associations have recently collaborated to align courses in computer science fundamentals to national education standards in hopes of more regularly folding in computer science instruction into the daily school schedule ("Code.org," 2018). It is, therefore, important to carefully consider and examine how engaging in computational problem solving affects both academic and cognitive outcomes.

Computer Science (CS) is an academic subject that is often viewed as intimidating and difficult to learn for many students. Using programming environments that reduce the cognitive load inherent in traditional programming languages due to complex and unfamiliar programming syntax is crucial for young children to access and develop computational thought processes (Kelleher & Pausch, 2003). Kafai and Burke (2013) note that the past decade has seen a rise in the number of introductory programming languages that make coding a more intuitive and personal process. The opportunities to cultivate the educational potential of learning to program are many, and the consequences are profound; however, many questions still remain in light of the existing literature on how computer-programming activities can affect educational outcomes.



The current study is one of the first in the field of educational research in computational thinking to utilize a control trial design to experimentally examine the relationship between participation in computer programming activities, problem solving, and academic achievement (particularly in the domain of mathematical problem solving) using standardized measures commonly found in the world of cognitive and academic assessment. The primary research questions being evaluated in this study are as follows:

1) Do youth who participate in programming activities demonstrate measurable changes in problem solving ability and creative thinking? And, 2) did they learn and apply computational thinking skills after participating in an introductory computer coding curriculum using a novice-oriented, graphically-based programming environment?

An intuitive link between mathematical problem solving ability and computer programming exists because programming utilizes principles of logic much like the use of inductive and deductive logic plays a primary role in mathematical problem solving, and it was hypothesized that children who engage in computer-programming activities will show significantly higher levels of mathematical problem solving ability and fluid reasoning ability as compared to those who do not engage in enriched computer-programming activities. Additionally, children who receive instruction in computer programming through a guided curriculum were hypothesized to demonstrate gains in programming conceptual knowledge and be able to apply this knowledge in a post-test of programming conceptual knowledge. Lastly, due to the open-ended, and limitless nature of lessons within the selected curriculum, measures of creativity are hypothesized to increase after completing the programming course.

## CHAPTER 2

### REVIEW OF THE LITERATURE

The increasingly complex and interconnected world of the 21<sup>st</sup> century is a double-edged sword that offers both incredible possibilities, and grave uncertainties. The exponential growth of computer processing capabilities in the twentieth century, combined with ever cheaper, more efficient data storage technologies (among many other advances in science and engineering) has impacted nearly every aspect of daily living, reaching even the most remote corners of the earth. From the smart phones that allow us to video chat with loved ones across the planet, to the vast electrical grid that connects power plants around the globe, there is a network of integrated systems and technologies continuously communicating, responding, and changing according to a complex set of logical rules and binary signals.

The sacrifices needed to develop such technologies have consumed time, energy, and natural resources on a scale never before witnessed by the human species. The scientific community unequivocally agrees that humankind has altered global environmental systems, and that the biodiversity of ecosystems both on a small and large scale is being threatened (Intergovernmental Panel on Climate Change, 2013). What society needs are creative minds that have the skillset and confidence to take on both the large and small-scale problems using the tools, strategies, and knowledge at their disposal. Of increasing relevance and importance is the ability to solve real world problems through the design and implementation of digital systems using principles of computer science (CS), a concept that has been termed computational thinking (CT). Articulating the difference between *education technology*, i.e., using computer

technology to learn about other subjects; *information technology*, i.e., the proper use of technologies by which people manipulate and share information; and *computer science*, i.e., the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society has been the focus of organizational initiatives to develop educational standards (Computer Science Teachers Association, 2011, 2017).

A comprehensive understanding of the intricacies and technical details of how the staggering number of digital systems and laws of nature interact and operate within the world is undoubtedly beyond the mental capacity for any single person; however, understanding the basic rules and languages that govern these systems is an achievable task that can be fostered through explicit instruction and collaborative experimentation. The short-term benefits of knowing how to use the language of computers to abstract and manipulate data could result in an employment opportunity among a variety of traditional industries and newly emerging fields. In addition, the movement toward a more digitally fluent global culture achieved through increased knowledge and participation in computational creation has the potential to solve some of humankind's most pressing issues.

Primary and secondary education is tasked with providing young citizens with the knowledge and opportunities to be informed and productive members of society who are capable of adapting to and influencing their natural and social environment. A crucial ingredient to the recipe for citizenship is the ability to critically understand and analyze information with objectivity, which is analogous to the understanding of science and the scientific method. A fundamental understanding of biological systems and physical laws

as well as the way humans interact both on a social and technological level to affect natural systems is an essential responsibility of all who aspire to positively influence the world around them. Moreover, providing students not only with knowledge, but more importantly, with the skills to think critically should be the ultimate goal of education, and is the very foundation of democracy (Bruner, 1960).

Modern civilization is the culmination of technological innovation and human ingenuity driven by environmental and economic forces, fueled by scientific discovery. Manipulating the environment to achieve desired ends and streamlining the process of realizing those ends is a hallmark of global economic and societal progress, and more broadly, humankind's adaptability and success as a species. The anthology of critical inventions and discoveries prior to and during the modern era is beyond the scope of the current review; however, within the last decade the pace of scientific discovery and technological innovation has been particularly accelerated as wireless communication and integrated computer networks have allowed individuals, businesses, and nations to more rapidly exchange information, services, and products.

The societal and economic dynamics of the information age are discussed at length in Castells' (1996) *The Rise of the Network Society*. The three-volume work thoroughly explores and describes the ways in which the instantaneous flow of information, currency, and cultural capital affect the lives of those living within its physical and virtual boundaries. Castells discusses the historical context of technological innovation and its relationship to the progression of civilization across time and space, while also laying the theoretical groundwork for how the new economy of information technology would evolve over time. Among many other things, he focuses specifically on

the importance of an educated portion of the population in contributing to societal and economic progression by affecting technological change as it relates mass media culture.

The economy of the information age has been in large part driven by the proliferation of ever-increasing means to quantify individual and group behavior through digital information. In a white paper prepared for the Computing Community Consortium committee of the Computing Research Association, Bryant, Katz, and Lazowksa (2008) describe the explosion of data within the previous decade as a result of technological developments in sensors, computer networks, data storage, cluster computer systems, cloud computing facilities, and data analysis algorithms. Bryant et al. (2008) call for increased investment in networking infrastructure, education, and research to address the existing limitations on utilizing this “big data” to its full potential.

As the sheer amount of data produced by an individual, or organization exponentially increases over time, the skills to analyze, transform, and interpret the data generated from the growing amount of digital activities inherent in daily life are becoming increasingly valuable (Lohr, 2012). The implications of this relatively rapid and dramatic shift in the amount of information available to governments, corporations, and individuals are such that a mere understanding of how to functionally navigate the digital world to obtain goods, services, and entertainment is now nearly an instinctive activity. Although being a user of digital devices makes up a large portion of individuals' interactions with computing devices, a movement beyond the mentality of simply being a user of digital technology in favor of being a participatory contributor is now more than ever being recognized as an important part of modern life, and an equally important educational opportunity (Kafai & Burke, 2013).

Despite the apparent importance of understanding fundamentals of computer science, and scientific knowledge in general, the state of science education in the United States continues to be the focus of criticism, even after over 30 years since the publication of the seminal report, “A Nation at Risk: The Imperative for Educational Reform” (US Department of Education, 1983). The perpetually bleak portrayal of K-12 science education in the US continues in part because of a relatively mediocre international ranking on the Program for International Student Assessment (PISA), in addition to unexceptional results from the science portions of the US National Assessment of Educational Progress (NAEP), which indicated that just 21 percent of high school seniors in the US were proficient in science knowledge (Fleischman, Hopstock, Pelczar, & Shelley, 2010). National initiatives specifically to support teachers and schools in the delivery of high quality Science, Technology, Engineering, and Math (STEM) education for their students have been initiated and continue to undergo refinement. In 2005, for example, the National Science Foundation (NSF) developed a new vision for science and engineering research and science education for 2020 (NSF, 2005). In 2010, the Computer Science Education Act, which aimed to provide grants to state educational agencies to strengthen CS education at the elementary and secondary level, was introduced in the United States House of Representatives in 2010 (H.R. 5929, 2010). In 2014, the STEM Education Act was introduced and passed in the house (H.R. 5031, 2014). This bill provided funding through the National Science Foundation (NSF) to research and development for STEM out-of-school learning and STEM learning environments, and research that advances the field of informal STEM.

Relatedly, in 2012, the National Research Council developed a comprehensive framework for K-12 science education in the United States, commonly referred to as the Next Generation Science Standards (NGSS), which focuses on the integration of core science concepts and practices into dynamic learning experiences across grade levels (National Research Council, 2012). Around the same time, the Association for Computing and Machinery (ACM), the Computer Science Teachers Association (CSTA), the International Society for Technology in Education (ISTE), and the National Science and Math Initiative (NSMI) along with advisors in the computing community (higher education faculty, researchers, and K-12 teachers) convened a task force to rethink how computer science education can be incorporated into K-12 classrooms. Based on computer science standards previously developed in 2003 and then revised by a CSTA task force (CSTA, 2011), these updated standards packaged computer science as a method of learning more traditional academic subjects while promoting 21<sup>st</sup> century competencies. The developers of these standards argue that computer science, as opposed to computer literacy, should be considered a core component of the general education curriculum because it bolsters critical thinking and problem solving skills. The CSTA task force released a newly revised version of the CS Standards at the 2017 CSTA Annual Conference, redeveloping computer science learning standards to specifically align not only with the NGSS framework, but also with Common Core State Standards, and the Partnership for 21<sup>st</sup> Century Skills: Essential Skills for Success guidelines (CSTA, 2017). The goal of both the NGSS and CS standards is to provide a framework for state and local education agencies to develop their own standards, and attempt to provide example activities and lessons designed to tap into various areas. It was based in

the assumption that educators must be skilled and competent enough to effectively incorporate learning activities designed to promote computational thinking skills at a more general curriculum level.

The term computational thinking (CT), popularized by Jeanette Wing in 2006, in its simplest form refers to the ability to understand how computers can be used to create things or solve problems. It involves abstracting data, thinking algorithmically, evaluation and generalization, modeling, and specific processes and perspectives. These skills are high-level cognitive processes that are often difficult for some teachers to understand (Grover & Pea, 2013). This is one reason why teachers have difficulty incorporating activities designed to promote CT skills into their instruction. In general, however, many researchers in the field of computer science in education feel that even very young children have the abilities to grasp CT concepts, and therefore, should be exposed to CT lessons and activities early in their education. This term has become subject to continuous practical redefinition and conceptual refinement since its introduction, and is discussed in more detail in a separate section below.

The educational setting has the potential to serve as an ideal medium for which to incorporate the NGSS and CS standards because of the growing importance in society and broad utility for skilled individuals with well-developed CS competencies across disciplines and industries; however, K-12 education has failed to tap into this potential. A 2010 report commissioned by the ACM and the CSTA found that despite emphasis of national, state, and local policy makers on the expansion of high quality STEM primary and secondary education, the number of schools offering introductory courses in computer science declined by 17 percent between 2005 and 2009, and schools offering



advanced courses in computer science declined by 35 percent in the same time period (Wilson, Sudol, Stephenson, & Stehlik, 2010). In a similar vein, a review of the existing literature surrounding the teaching of computer coding at the elementary level indicates that although there is a call to incorporate coding in the US elementary curriculum, few schools are actually implementing such activities (Pinkston, 2015).

A renewed public interest in computer coding fueled by increased recognition of the practical importance of computational thinking skills in the workforce, and the global propagation of education-based organizations whose mission is to develop innovative problem-solvers and promote CS for all, is opening the door for computer science to be woven into aspects of K-12 education beyond its current focus on education technology. Recently, The White House has called for a “Computer Science for All” initiative that plans on dramatically increasing funding for states to expand K-12 computer science education by training teachers, expanding access to high-quality instructional materials, and building effective regional partnerships (The White House, 2016). Most recently, development of a revamped high-school level AP Computer Science Principles course was completed with a focus on lessons and projects designed to explore computational concepts, processes, and practices, while also more clearly laying out learning objectives and assessment methods (The College Board, 2017). The exciting future of computer science education, however, should not overlook its past, and in the following section, some of the milestones and relevant research related to computers in education are presented and discussed.

## Historical Developments in Educational Technology

The history of computing technology and computer science in education would not be complete without mention of the work of Seymour Papert, who most notably authored *Mindstorms: Children, Computers, and Powerful Ideas* (1980). In this book he argued that children have the capability of understanding how to use computers on a complex level, and that learning to use computers in this way can change how they think about the world. He rooted this idea in constructivist learning theory, which in its simplest form suggests that individuals construct meaning and build knowledge in relation to their experiences and ideas. This epistemological philosophy is evident in the writings and theories presented by such influential psychologists as Jean Piaget, who focused on how individuals construct knowledge and meaning from a developmental perspective (Piaget, 1962); and Lev Vygotsky, who incorporated socio-cultural and historical contexts to explain how individuals construct knowledge of the world to develop higher-level cognitive processes such as problem solving ability (Vygotsky, 1978). Papert, however, elaborated on the constructivist perspective on learning, suggesting that the context in which individuals most effectively build knowledge structures occurs when they are actively engaged in constructing a public entity that is meant to be shared with others. The term he used to describe this assertion was *constructionism* (Papert, 1991).

The seeds of constructionism had been planted in Papert's mind long before the term was coined with the development of the Logo programming language in 1968 at the Massachusetts Institute of Technology (MIT) based technology company, Bolt, Beranek, and Newman (BBN) by Seymour Papert, Wallace Feurzeig, and Daniel Bobrow. The

Logo programming language was designed as a tool for learning through simulations, multimedia presentations, and games in mathematics, language, music, robotics, telecommunications, and science (“Logo History”, 2015). The Logo programming environment utilizes a turtle, or “sprite,” that can be programmed to move in a virtual space to create and manipulate graphics, geometrical shapes, and designs. As users become more proficient in Logo, they are able to execute more complex series of commands while receiving immediate visual feedback on their programs. Papert (1980) claimed that children learn to use the Logo turtle as an “object-to-think-with” (p. 11) allowing them to link their internal representations of the virtual world of the Logo turtle with the physical world in which they inhabit. He claimed that all learners regardless of age or ability could learn to use computers to construct knowledge about the world, and he profoundly influenced a generation of scientists, educators, and students with his philosophy. The popularity of Logo as a programming language waned in the 1990s, but was renewed as variations of the language that allowed for more functionality, interactivity, and capability were developed under direction of Mitchell Resnick at MIT and through collaboration with Uri Wilensky of Northwestern University, e.g., StarLogo, NetLogo, LEGO Logo (Hayes & Games, 2008; “Logo History,” 2015).

As personal computers became more widespread in the subsequent decades following the development of Logo, and in tandem with the popularity of *Mindstorms*, educators and computer science researchers became interested in how Logo could be used to not only introduce students to the burgeoning world of computer science, but also how it could be used to enhance specific academic skills. A library database search of scholarly and peer-reviewed journal entries with the term “Logo” in the title and the

terms “learning” and “programming” in the abstract yielded 37 articles from 1980 to 2016 peaking in 1986. A more thorough search including search terms describing other programming languages of the era like BASIC and Pascal, which were programming languages developed with a similar goal, i.e., to introduce computer programming to children, yielded a larger return of articles; however, the wave-like trend beginning in the mid to late 1980s and lasting until the early 1990s for the emergence of studies involving these programming languages remained the same. Many of the articles returned in the original search investigated the relationship of programming with Logo to mathematical understanding of geometric or algebraic concepts (Clements & Sarama, 1995; Feurzeig, 1986; Noss, 1986; Olive, 1991; Valente, 2003) or more general problem solving abilities (Au & Leung, 1991; Battista & Clements, 1986; DeCorte, 1992; Howard, Watson, & Allen, 1993; Khasawneh, 2009; Pardamean & Evelyn, 2014; Poulin-Dubois, McGilly, & Shultz, 1989; Suomala, 1996). Another subset of research with Logo programming investigated *how* children learn while using Logo through various cognitive perspectives (Geva & Cohen, 1987; Gibbons, 1995; Mayer & Fay, 1987; Olive, 1991; Wilson, Mundy-Castle, & Sibanda, 1991; Yelland, 1995) or instructional methodologies (Emihovich & Miller, 1988; Fay & Mayer, 1994; Hoyles, Sutherland, & Evans, 1986; Lin, Li, Ho, & Li, 2007; Littlefield, Delcios, Victor, Bransford, Clayton, & Franks, 1989; Sutherland, 1993). Finally, one study evaluated how learning to program in Logo affected a measure of creativity (Clements, 1986).

Of the studies identified in the limited literature search mentioned above, many report learning gains across measures when students engaged in programming with Logo (Au & Leung, 1991; DeCorte, 1992; Emihovich & Miller, 1988; Mayer & Fay, 1987;

Pardamean & Evelyn, 2014; Suomala, 1996). On the contrary, other researchers found that without sufficient instructional support, Papert's ubiquitous belief that any child could construct meaningful knowledge about the world through exploration and creation in a virtual world (mainly through Logo) was not supported by improved problem solving ability or mathematical achievement (Battista & Clements, 1986; Clements, 1986; Cohen, 1987; Littlefield et al. 1989; Palumbo, 1990; Pea & Kurland, 1984; Wilson et al., 1991).

One such study challenging Papert's claims that all students are capable of learning how to program with computers involved preschool children who were instructed on how to use the Logo environment for 45 minutes per week for eight months (Vaiyda, 1985). Fourteen preschool children ranging from 55 to 65 months in age were assessed across the following four variables: 1) field dependence-independence (a measure of cognitive style), 2) creativity, 3) mathematical ability, and 4) computer or computer-related experiences in the home and outside the home. Children were categorized into three groups corresponding to their ability to use Logo effectively as determined through structured observation. None of the measured variables significantly differed across groups; however, all of the children in the group that demonstrated the most advanced understanding of Logo had video games in their home and played arcade games, as opposed to three out of the ten children in the other less advanced Logo programming groups. This early attempt to teach preschool-age children how to use Logo provided some support that young children could effectively interact and use functions of Logo; however, the relatively low dosage of the treatment, i.e., 45 minutes per week, combined with the extremely small sample size ( $n = 14$ ) may not have been enough to produce or detect changes across the measured dimensions.

In another study, Battista and Clements (1986) tested whether groups of fourth and sixth grade students who participated in either 1) computer-aided instruction (CAI) using programs designed in a game-like format intended to teach specific skills, e.g., *Rocky's Boots* from The Learning Company or *Thinking With Ink* from the Minnesota Educational Computing Consortium; or 2) used Logo Turtle graphics after whole-class instruction, performed differently on problem solving and math achievement measured by responses to researcher-created word problems categorized either as procedural and conceptual problems, or executive processing levels of problems solving. Students participated in two 40-minute sessions per week for a total of 42 sessions across the school year. The researchers found no significant differences between students who used Logo, CAI, or control groups from pre to post-test measures of procedural and conceptual aspects of problem solving; however, the Logo group demonstrated significantly greater gains from pre-test to post-test than either CAI or control groups in metacognitive aspects of problem solving, i.e., deciding on the nature of the problem, choosing a solution strategy, selecting a mental representation, and monitoring solution processes. The authors concluded that there was no evidence that computer-aided activities improved students' procedural or conceptual knowledge of math or ability to solve problems, but may improve executive-level problem solving skills.

In a small scale study, Kurland and Pea (1985) investigated how a group of eleven and twelve-year-old children ( $n = 7$ ) verbally explained how recursive Logo programs functioned after roughly fifty hours of classroom programming time in Logo over two years. Classroom programming time consisted of exploratory lessons in Logo, and students were provided direct instruction in iteration and recursion. Recursion is a

cognitively complex concept important in computer science that allows a function to call itself within the program text, enabling an infinite number of solutions to a given problem using simplistic commands arranged and defined appropriately. Students were asked to look at short Logo shape-drawing programs of varying levels of complexity, then to give a verbal description of how the program would work, and finally, to hand draw how the program would run. The authors found that children demonstrated significant difficulty explaining recursive functions, and often misinterpreted the context, assigned intention to the Logo turtle, used incorrect concepts, e.g., looping, or attributed natural language meanings to the computer code.

Although the majority of the studies described above occurred in the mid to late 1980s, some researchers have called for reevaluation of this era in the literature due to its importance for the future of computer coding in education (Grover & Pea, 2013). In reality, however, the field moved in a different direction partly as a result of conflicting results from a number of studies investigating the potential educational and cognitive benefit of programming with Logo, combined with advancements in computer processing capabilities and software portability, i.e., floppy disks and CD-ROMS. Computer technology in the educational environment began to gravitate away from the programming-for-all philosophy championed by Papert through Logo, and toward the use of educational games as an emerging industry of children's software began to dominate the landscape in the mid to late 1990s.

Valente (2003) notes that in the 1980s, computers in schools were very simple machines that were relegated to one of two roles, i.e., simple drill and practice machines to teach specific content through tutorials, simulations, or games, *or* for programming

activities primarily using Logo. Ito (2008) describes the cultural history of educational software aimed at elementary students, tracing the development of children's software, learning games, or "edutainment" as it progressed through the 1980s and 1990s. He describes the shift from drill-and-practice CAI systems to software that drew from aspects of the growing video and arcade game industry, e.g., *Number Munchers*, *Math Blasters*, *Oregon Tail*, *Reader Rabbit*, *KidPix*, and *Where in the world Is Carmen Sandiego?* He further decomposes educational software into three strands, each associated with either behaviorist, play-centric, or constructivist educational philosophies. The academic strand typically embedded academic mini-games within a larger role-playing or action-based scenario, e.g., *Math Blasters*, choosing to focus more on curricular alignment and behaviorist principles of positive reinforcement for task completion rather than innovative game-play. The entertainment strand primarily focused on play-centric "click-and-explore" interactive environments, e.g. *Myst*, with some subtle academic content embedded within the environment as opposed to overt academic challenges in order to obtain a reward as part of game play. The constructive strand rested heavily on Papert's ideology and provided children with toolkits to create computer-based programs or simulations, e.g., *SimCity*. These types of software were intended to promote technical empowerment, i.e., "the ability to translate authorial agency into a media form" (Ito, 2008, p. 101). Although many of these gaming titles were marketed to parents, educators and schools began to buy in to some of the claims that children could learn important academic knowledge through gameplay. The fact that children were entertained and motivated by these educational games was a strong selling point for their



adoption as a part of computer education in schools and in homes across the United States and elsewhere in the world.

The changing technological landscape continued to influence computers in education through the late 1990s, and an increasing emphasis was placed on computer literacy. As computers became cheaper, smaller, and loaded with more modern and easy-to-use operating systems, a large amount of federal funding was set aside to computerize schools. For example, the 1996 Technology Literacy Challenge Fund allocated two billion dollars for schools to provide training, resources, and infrastructure to “connect all classrooms in America to the information superhighway” (Riley, Kunin, Smith, & Roberts, 1996). The computerization of American classrooms dramatically reduced the ratio of computing machines to students, opening new doors for how computers could be used in schools, and tasked both educators and students alike to develop computer literacy skills, i.e., possess the knowledge and skill to functionally navigate through a computer interface, use word-processing software, and explore the internet.

Specifically, a virtual hands-on approach to learning and research became possible through developments in word-processing and presentation software, e.g., Microsoft Word and PowerPoint, through the incorporation of smart-board technology as a learning tool to enhance classroom instruction, and through the availability of various forms of media with increasingly powerful web browsers. The World Wide Web became more complex and user-friendly, and collaborative classroom environments that integrated the use of dynamic classroom blogs and other digital learning communities with traditional instructional methods emerged as a common practice in secondary and higher education. This “blended,” or mixed, classroom wherein students submit work,

receive feedback, and critique others' work continues to alter the manner in which instruction is delivered and learning is assessed.

The movement toward computer literacy in the 1990s and 2000s, albeit a necessary progression, overshadowed the fundamental tenets of computer science that had been so much a part of the early endeavors in computer programming in education during the 1980s with Logo and other early programming languages such as BASIC and Pascal. Some schools may offer an additional computer technology class where students are educated on how to sift through the immense amount of information available to them on the Internet for research purposes, and use various software to create multimedia presentations and projects, but rarely do these supplemental classes introduce students to CS and its applications, or exist uniformly across districts and regions that have varying financial and logistical resources. A transformation and return to CS fundamentals appears to have occurred in recent years, as a resurgence of computer coding, fueled by national initiatives and non-profit organizations, has captured the attention of important and influential stakeholders ranging from parents to policy-makers. Countries around the world have recently adopted CS as part of their school curricula, e.g, the UK and Denmark (Caspersen & Nowack, 2013); Russia, South Africa, and Israel (Zur-Bargary, 2012); New Zealand (Bell, Andreae, & Robins, 2012) and South Korea (Choi, An, & Lee, 2015) to name a few.

In the book *Connected Code*, Kafai and Burke (2014) describe the return of interest in teaching children coding as a result of increased recognition that thinking like a computer scientist is an important step in solving real-world problems, designing useful systems, and succeeding across disciplines. The authors explore the aspects of

programming that are appealing to children, the contexts in which children use programming, and how children learn to program specifically using the *Scratch* programming environment (an extension of the Logo programming language). By focusing on: a) how computer coding can be applied to make things, e.g., video games and digital stories; b) the ways in which it is not just an individual venture but also increasingly a social activity; c) how repurposing or “remixing” computer code affects skill acquisition; and d) how code can be used in beyond the computer screen, e.g., robotics, the authors explain how computer coding has once again become a topic of interest in education, and why it is now as important as ever.

The introduction of the Scratch programming environment has influenced how children learn to code, how educators can utilize the platform to teach CS concepts and academic content, and how researchers can better understand what it means to think computationally. Scratch is a simplified, easy to use, and powerful introductory programming environment that was born out of the MIT-based Lifelong Kindergarten research group, extending the development of Logo Mindstorms for the Lego Company to create a playful programming language that, much like snapping Lego bricks together, became a visually-based, drag-and-drop, block-command environment (Resnick, Kafai, & Maeda, 2003). Scratch follows Papert’s (1980) guidelines to a successful programming language and environment intended for use with young learners, i.e., a low floor, a high ceiling, and wide walls. In other words, the syntax of the language should allow students with no background whatsoever in computer science the ability to write and understand programs (low floor), while also allowing users to fine tune their skills to create infinitely complex programs to solve increasingly complicated problems as their familiarity and

skills increase (high ceiling), and support many different types of projects, so that users who possess a wide variety of interests and skillsets can interact and create on a personal level (wide walls). The Scratch environment encourages “tinkering” to create personalized digital media ranging anywhere from animations to games. Users arrange various command blocks to create stacks of code that can include loops, conditionals, variables, data structures, and even user-created functions (Resnick et al., 2009). The Scratch website ([www.scratch.mit.edu](http://www.scratch.mit.edu)) serves as a community hub for member-created projects for others to view, interact with, and “remix.” Due to its easy to use and visually pleasing design and interface, Scratch has become popular among children, teachers, and researchers alike.

### Computational Thinking as a 21<sup>st</sup> Century Literacy

Coding is a critical skill, and has even been described as a new literacy for all children (diSessa, 2000; Rushkoff, 2011), but what does “coding” mean? Essentially, coding is applying the language of computers to achieve a desired result. The term has risen in popularity in recent years (Kafai & Burke, 2013), and to understand what it truly means to code, one must also understand what it means to think computationally.

Computational thinking (CT) in its modern conceptualization, was popularized after Jeannette Wing published an article using and describing the term in the March 2006 edition of the *Communications of the ACM*. Wing (2006) defined CT as designing systems for more effective problem solving with computers. The Royal Society (2012) describes the essence of CT by stating that it is “the process of recognizing aspects of computation in the world that surrounds us, and applying tools and techniques from Computer Science to understand and reason about both natural and artificial systems and

processes” (p. 29). Wing (2008) elaborated on her original definition by suggesting that CT is the automation of abstraction; moreover, it is a process of design that attempts to answer the questions, what are computers better at, and what are humans better at? In an article featured in *The Link* magazine (a magazine of the Carnegie Mellon University School of Computer Science) in March of 2011, Wing was inspired by electronic discussions among colleagues to addend her definition of computational thinking to encompass the following: “Computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent” (Wing, 2011). One of those colleagues, Al Aho, defines CT as the thought processes involved in formulating problems so that their solutions can be represented as computational steps and algorithms (Aho, 2012).

Brennan and Resnick (2012) provide a framework of analyzing computational thinking across the following three dimensions: “computational concepts,” i.e., sequences, loops, parallelism, events, conditionals, and data structures; “computational practices,” i.e., being incremental, reusing and remixing, testing and debugging, and modularizing and abstracting; and “computational perspectives,” i.e., expressing, connecting, and questioning. The last dimension refers to the way that designers view and engage with digital media. This operational definition for CT has subsequently been used to categorize the CT literature by various researchers (Kafai & Burke, 2016; Lye & Koh, 2014). Grover and Pea (2013) discuss computational thinking using Brennan and Resnick’s (2012) three-pronged operational definition and provide an overview of how computational thinking in education in the United States has yet to be realized in the K-

12 arena. The authors conclude with a call for research investigating developmental expectations and trajectories associated with aspects of computational thinking.

Likewise, Lye and Koh (2014) used Brennan and Resnick's (2012) three-pronged definition of computational thinking to analyze 27 studies of computer programming activities carried out in K-12 and Higher Education settings. The authors included only articles published in peer-reviewed journals in their search criteria. They identified nine studies in the literature that were carried out in the K-12 environment, and concluded their review by highlighting the importance of studying the developmental trajectory of computational thought processes in young children, specifically calling for examination of learning outcomes outside of the computational concepts component of Brennan and Resnick's (2012) operational definition of CT. They also point out the need for more studies in the classroom environment, as opposed to after-school programs.

In response to these calls for developmental clarification, Selby, Dorling, and Woollard (2014), working for the Computing at School (CAS) organization in the UK, attempted to delineate the definition of CT by developing the "Computing Progression Pathways" framework for the assessment of CT skills. This endeavor was undertaken to provide standards or the National Curriculum Program for the Department of Education in the UK across CT areas and over a range of developmental stages. These researchers used Selby and Woollard's (2013) conceptualization of CT skills, which was developed by establishing the following criteria for CT thought processes through a literature review of CT definitions: abstraction, decomposition, algorithmic design, evaluation, and generalization. They then categorized each stage of development across content categories according to these CT thought processes. The framework specifically outlines

developmental expectations across content categories of the CAS curriculum, i.e., algorithms, programming and development, data and data representation, hardware and processing, communication and networks, and information technology, and provides narrative descriptions for various age ranges across CT concepts, i.e., algorithmic thinking, evaluation, decomposition, and generalization across age-bands (Computing at School, 2012). Similarly, The CS standards released by the CSTA in 2017 is organized broadly by age level, as well as CS concepts and subconcepts, with associated practices across age levels and subconcept areas (CSTA, 2017). The hope is that by providing age-based expectations for CT skills and CS concepts, researchers and educators will be more effectively able to teach students and measure their student learning. These frameworks, however, may need further validation research to solidify the normative developmental trajectory across ages, but regardless of whether or not they are reliable and valid standards evidenced through empirical support, they may serve as useful tools for more inexperienced instructors to better measure and incorporate CT in their lessons and activities.

#### How Can Educators Best Support the Development of CT Skills?

Researchers and professional organizations, e.g., the CSTA, have urged the field of computer science education to develop practical ways in which educators can incorporate the tenets of CT into their daily classroom activities. Barr and Stephenson (2011) outline core computational thinking concepts and corresponding ways that these concepts can be represented in various academic subjects. For example, they suggest that the CS principle of abstraction, i.e., the use of procedures to encapsulate a set of often repeated commands that perform a function (conditionals, loops, recursion, etc.) is

analogous to summarizing facts, and deducing conclusions from facts in the academic area of social studies. The development of transposable CS standards that are easy to align with academic content is the beginning of fostering CT skills in K-12 education, and moving forward, the challenge to proponents of CT in the K-12 curriculum will be to translate these standards into understandable, easy-to-implement activities across grade levels and subjects even for teachers who do not consider themselves skilled or knowledgeable in computer science. Efforts to integrate CT into core curricula have been made by professional associations, e.g., the CSTA, ISTE, and non-profit organizations like Code.org, Computing at School, Globaloria, and Shodor, which provide materials and instruction relating to computational science (“Shodor,” 2016), as well as corporate entities, e.g., Google’s Computational Thinking website ([www.google.com/edu/ect](http://www.google.com/edu/ect)) that offers videos explaining what CT is, and resources for educators to facilitate the integration of CT in the classroom. An innovative group based in New Zealand has even developed ways to introduce CT skills without the use of computers. The Computer Science Unplugged website offers a collection of free learning activities that teach concepts such as binary numbers, algorithms, and data compression through games and puzzles using easily producible classroom materials ([www.csunplugged.org](http://www.csunplugged.org)). The resources to teach CT in the K-12 environment exist, but implementing them on the ground is a challenge that may still take time and energy to overcome.

Despite efforts to integrate CT in the classroom, students in the K-12 arena predominantly engage in computer programming activities in out-of-school camps or clubs (Lye & Koh, 2014). While informal learning settings have been shown to enhance scientific reasoning ability (Gerber, Cavallo, & Marek, 2001), these environments pose



difficulties for educators, as both student and teacher motivation toward learning objectives may differ from more formal settings (Kisiel, 2005; Lucas, 2000). While many studies of out-of-school time (OST) enrichment programs demonstrate some positive impacts across social and academic areas, these environments are associated with low student attendance, and difficulty implementing structured activities and tasks (Dynarski, James-Burdumy, Moore, Rosenberg, Deke, & Mansfield, 2004). Fostering CT skills through explicit instruction and supplemental activities during school hours may, therefore, be met with greater student engagement and learning outcomes.

Although some aspects of CT can be taught to early elementary students, especially through the CS Unplugged initiatives that require little to no familiarity with computer code in the traditional sense, the middle school years are particularly important for students to develop and grow the cognitive and social skills needed for future educational endeavors, especially within the STEM fields (Tai, Liu, Maltese, & Fan, 2006). One reason that many of the studies in the field of computer science education have focused on the middle school years (ages 11-13) is the notion that abstract thought emerges during this developmental period (Piaget, 1936). Being able to conceptualize information in an abstract way is a central concept in computer science and this idea is even captured in the very definition of computational thinking (CSTA, 2011). This is not to say that children below this age band are incapable of learning CT skills, and in fact there is evidence that some children as young as seven can benefit from learning about computational concepts (Li, 2010). In an exploratory case study Fessakis, Gouli, and Mavroudi (2013) used a series of Logo-based activities with kindergartners in tandem with an interactive whiteboard to input navigational commands for a ladybug sprite to

hide under a leaf. Results indicated that the children actively participated and enjoyed solving problems through planning and trial-and-error methods, and their mathematical skills related to direction and orientation seemed to improve through participation in the activities; however, the authors provided no quantitative measures to support their claims. It is, therefore, important for educators to consider introducing young children to the world of computer science through hands-on computerized and non-computerized lessons at an early age, and continue to foster the development of CT skills throughout early to middle childhood.

### *Games and Game-Based Learning*

The most popular way in which proponents of computational thinking have set out to teach these skills to young learners is through designing games (Wu & Richards, 2011). The cognitive benefits of playing games was recognized long before computers became an integral part of daily life, as evidenced in Piaget's (1951) work that explored the developmental importance of game play as a way for children to refine and apply their understanding of rules. As one of the founders of constructivism, Piaget viewed the construction of games one of the foremost methods of game play. Kafai and Resnick (1996) use the constructionism perspective to frame a discussion on the impact of computational technologies on children's learning, education, and knowledge. Specifically, they focus on how creating computer games and other projects, as opposed to merely playing computer games, can influence how children learn through design. Gee (2003) wrote extensively about the potential of both playing and designing video games as an avenue to promote learning and literacy. He argued that giving students the chance to personalize their own game could be a powerful way to engage students, and instill

within them a sense of pride and accomplishment that facilitates learning. He went on to outline 36 educational principles that could be cultivated in the design and play of video games. Squire (2006) described how the experience of playing games affects the ways players think about history, physics, and academics in general. Furthermore, he argued that making games through computer programming activities would be an even more powerful way of affecting the ways in which players learn and think about the world. Squire (2006) has gone as far to say that traditional educational pedagogies should adjust the delivery of academic content to match the changing times; moreover, that engaging students to think critically about the world can and should be accomplished through gameplay and game design.

Hayes and Games (2008) provide a review of the various novice-oriented computer software for designing and making games available at the time, as well as the instructional strategies intended to engage young learners in making games. The researchers separate lines of research into the following four themes: game creation as a way to teach programming tools or concepts, game creation as a way to bring more girls and women into the field of computer science, using games to teach academic content, and game creation to learn specifically about how games are made.

The enormous success of the video game software industry in the 1990s, when software publishing companies grew to become some of the most successful businesses in history, changed the ways computers were used in the classroom (Cuban, 2001). In attempts to capitalize on youth's fascination with video games, researchers in the field of computer science education began to develop more appealing interfaces with game-making functionality to teach computer programming concepts. Soon, they found that

exposure to programming through an introductory, guided, game-based curriculum increased knowledge and confidence in programming concepts, which is an important factor for students to continue their computational explorations beyond the classroom or research setting (Al Bow, Austin, Edgington, Fajardo, Fishburn, Lara, Leutenegger, & Meyer, 2009). Nearly every study in the current review in some way used an aspect of computer game design or game-like activity as part of its instructional strategy to teach fundamental computer science concepts, partly due to the influence of Papert's (1991) idea of constructionism. For a thorough review of the educational benefits of student-created games, see Kafai and Burke's (2016) synthesis of 55 studies in the K-12 environment that summarizes and categorizes empirical endeavors across Brennan and Resnick's (2012) operational definition of CT, i.e., computational concepts, computational strategies, and computational perspectives.

Educators and researchers generally agree that youth may be able to gain valuable skills across a range of educational competencies through both game design and game play, but the development of CT skills, specifically through game design, is not purely individual pursuit, and in fact can be enhanced by encouraging or requiring youth to work collaboratively, and provide feedback regarding one another's work.

#### *Peer Critique and Collaboration*

Involving peers in the development and critique of programming processes and products has been used as one way to increase learning of programming concepts. For example, paired programming has been a common practice in university settings, and has also been introduced into the K-12 environment as an instructional strategy to help students learn programming skills together (Werner, Hanks, & McDowell, 2004).

Peer feedback has also been linked to improved student outcomes. For example, Hwang, Hung, and Chen (2014) separated students into treatment and control groups differing in whether students received feedback from their peers about their projects. Students took a pre-test designed to measure background content knowledge, learning motivation, and problem solving skills before participating in a computer game development course. These 167 Taiwanese 6<sup>th</sup> grade students worked to develop computer games using *Kodu*, a software developed by Microsoft, as part of an environmental science unit on the effects of global warming. Fifty-minute long game design and development sessions were held once a week for ten weeks and students in the treatment group were allowed to give feedback to their peers on the enjoyment, appearance, completeness, accuracy, and relevance of their game design while students in the control group received no feedback from peers. At the end of the ten weeks, students took a post-test and the treatment group answered open-ended questions. Students in the treatment condition showed significantly greater ratings of learning achievement, problem solving skills, learning motivation, in-depth thinking, and creativity. The authors conclude that peer-based assessment can enhance that student's learning achievements and problem solving skills. Also, students in the reviewing treatment group reported higher levels of enjoyment, which indicates that peer-based assessment can be used as a tool to engage students in game development activities.

In another study, Su, Yang, Hwang, Huang, and Tern (2014) examined how peer feedback facilitated by a digital tool to make annotations to student projects in the Scratch programming environment, as well as differing pedagogical strategies, impacted problem solving and programming conceptual knowledge. Four classes of students

totaling 135 sixth grade Taiwanese students participated in two Scratch units over six weeks in various conditions of instructional strategies and differing levels of peer feedback. Results showed that students who critiqued other students' annotations of their thoughts throughout the computer programming activities in Scratch combined with explicit instruction in solving programming problems in a stepwise manner showed increased understanding of programming concepts as measured by a criterion referenced, instructor-created test of programming conceptual knowledge that consisted of a combination of multiple choice and project-based items.

Other more qualitative studies have also indicated that programming is learned best when it is learned in the context of a community environment where members are able to give and receive feedback on their programmed artifacts. For example, Baytak and Land (2011) and Werner, Denner, and Campe (2014) incorporated peer feedback and programming in pairs within their studies and reported that students demonstrated learning gains in the domains of computational thinking, and reported greater levels of understanding and enjoyment of programming-related activities.

#### What Can Students Learn While Participating in Programming Activities?

##### *Programming Conceptual Knowledge*

The most obvious learning outcome of participating in any type of activity that engages individuals with core programming concepts would be an increase in knowledge of said concepts. A large number of studies have investigated whether young learners actually do learn about the fundamentals of computer science and the results generally support that some degree of programming conceptual knowledge is attained after engaging with computer programming tools. In one example, Maloney, Peppler, Kafai,

Resnick, and Rusk (2008) introduced Scratch in an urban community center located in an impoverished neighborhood in Los Angeles, CA. The authors qualitatively describe the results of a yearlong observation period accompanied by descriptive statistics regarding youth-created projects; however, there was no control group for comparison of learning outcomes of youth not engaging in computer programming activities. The researchers took a hands-off approach to teaching Scratch, choosing to make research assistants available to answer student's questions rather than providing direct instruction on the functionality and features of Scratch. Although the presence of researchers in and of itself may have impacted youths' behavior and motivation to learn Scratch, the authors found that a culture of computer programming seemed to emerge, and that by creating projects in Scratch, youth demonstrated understanding of core computer programming concepts. The study not only highlights the collaborative and creative nature of the Scratch environment, but also the supportive and assistive aspect of the community center in relation to initial interests in pursuing Scratch projects. The researchers' assessment of programming knowledge, however, did not go beyond qualitative description so it remains unclear as to whether or not students actually understood the concepts they were using in their projects.

In one study that investigated how children learned programming conceptual knowledge, Baytak and Land (2011) used computer game development as a peer tutoring experience to bolster fifth grade students' computational thought processes. The researchers asked fifth graders to use Scratch to design computer games that were to be used to teach second grade students about environmental science. The study took place over the course of 21 sessions that included planning, design and development, and

testing phases. A science teacher was present during all sessions to answer questions, and students were encouraged to collaborate with one another. In order to assess students' knowledge of programming concepts, the researchers coded command blocks of each student's game into the following categories: statements, Boolean expressions, conditionals, loops, variables, threads, and events. The authors present narrative and case studies describing the student-created games and the game development process. They conclude that creating computer games within an academic context is a dynamic learning process that involves goal setting, information seeking, and problem solving through inter-student collaboration, and that a visually-based software environment (as opposed to a more traditional text-based programming environment) can help elementary students access and understand complex programming skills; however, the authors point out that the results were more exploratory and descriptive, rather than conclusive.

While the above studies were more qualitative in nature, Denner, Werner, and Ortiz (2012) investigated computer games created by 6<sup>th</sup> grade girls in an after-school program over the course of three months by a mixture of quantitative and qualitative approaches to understand whether students who used various types of code actually learned the corresponding CT concepts. Each participant or pair of participants created several games using the Stagecast Creator programming software, a novice-oriented, graphically-based, introductory programming environment with functions corresponding to foundational CT concepts and algorithmic thinking, e.g., objects and inheritance, methods, events, and code documentation. In order to determine the percentage of games that included various aspects of CT concepts, the researchers analyzed each game across 24 computer-code categories separated into three broad computer science competencies,



i.e., programming, code organization and documentation, and designing for usability. Results indicated that the youth did not use more complex programming features while creating their games, despite qualitative descriptions of high interest in the activity. The authors conclude that additional instructional support is needed in order to engage children of this age in more complex coding methodology. Whether these youth, or others engaged in similar computer programming activities, actually learned the CT skills they demonstrated by the incorporation of specific codes and sequences in their projects is subject for debate, however.

In a more recent study offering further mixed evidence to support the notion that although students may demonstrate interest, engagement, and completion of computer programming tasks, they may not necessarily understand the underlying concepts and be able to transfer this knowledge to a novel environment. Grover, Pea, and Cooper (2015) investigated whether student knowledge of algorithmic concepts, i.e., serial execution, looping constructs, and conditional logic, transferred to text-based programming languages after using Scratch in a design-based curriculum using both a face-to-face and blended classroom with children ranging in age from 11 to 14 years in a public school setting. The researchers used weekly quizzes and a pre/post-summative test to evaluate whether students were able to understand programming concepts and found that student knowledge of basic algorithmic flow of control in computational solutions was increased from pre to post-test; however, students demonstrated more difficulty understanding loops and variables. The knowledge transfer test was conducted in a novel, text-based programming language and focused heavily on loops and variables; thus, student performance did not show substantial understanding of these concepts. Additionally, the

researchers noted that student performance on programming conceptual knowledge tests did not differ significantly whether students were in face-to-face classroom or blended classroom condition.

### *Problem Solving Skills*

Extensive research and theorizing about the relationship of problem solving to cognition suggests that the process of solving problems is a fundamental characteristic of thinking (Mayer, 1977; Sternberg, 1994), and although discussion of this research is beyond the scope of this review, problem solving can generally be broken down into the following domains: knowledge representation, conceptual categorization, deductive reasoning, and inductive reasoning (Sternberg, 1994). One of the most widely touted claims about participation in computer programming is that it has the potential to improve general problem solving abilities. The supposition is that instructing a computer program to enact a set of rules to achieve a desired function is in itself a problem solving process, and that engaging in such activities can generalize more broadly to enhance problem solving ability.

In an early attempt to answer the question as to whether computer programming positively influenced cognitive outcomes, which served as an over-arching category and primarily included various measures of problem solving skills, a meta-analysis conducted by Liao and Bright (1991) analyzed the learning outcomes of 65 studies involving computer programming in education, of which 89 percent reported positive effect sizes, resulting in an overall moderate grand mean effect size (0.41). One issue with this analysis, as is the case with all meta-analyses, was that the authors considered the assortment of student learning measures in the same way, i.e., as uniformly standard

measures of cognitive skill determined by tests at the end of programming instruction. Although they concluded that computer programming can lead to improved student learning, mainly in reasoning skills, logical thinking, planning skills, and general problem solving skills, a more thorough understanding of how the studies included in the analysis measured these cognitive outcomes, and their validity is warranted in order to make such claims.

Some studies have looked at incorporating explicit modeling or teaching of problem solving strategies directly could affect their experiences in computer programming. For example, Akcaoglu and Koehler (2014) chose the Kodu programming environment in an after-school setting and used a computer game-design curriculum that was paired with instruction that explicitly taught the steps of problem solving as described in previous literature the following four-step process: *representing*, i.e., understanding the problem, *planning*, i.e., devising a solution by decomposing the problem, *executing*, i.e., putting the plan into action, and *evaluating*, i.e., checking to see if the plan resulted in achieving the goal (Jonassen, 2004; Polya, 1957). The researchers measured performance on questions from the Program for International Student Assessment (PISA) designed to measure students' skill at solving the following three problem types: systems analysis, decision-making, and troubleshooting. Results indicated that participants in the experimental group significantly outperformed those in the control group who did not receive the game-based design instruction, leading them to conclude that game-design through computer programming can improve problem solving skills.

There have been a number of studies in the field of computer science and education investigating whether or not problem solving skills are indeed enhanced

through computer programming activities, and while there is evidence to support the claim, questions remain for how problem solving is measured, and how to best facilitate learning. To understand how computer programming activities relate to problem solving skills, it is important to keep in mind that in order to be successful with the former, a prerequisite skill level in the latter is necessary. Without sufficient scaffolding in the problem solving process, learners may not be able to make the conceptual leaps needed in key CS areas and CT skills, and thus, experience success with their creations. In this way, a bidirectional relationship may exist between problem solving abilities and computer programming, and by teaching students about how to think about thinking, they may be better able to solve problems, and achieve their goals in computer programming activities more successfully.

### *Metacognitive Skills*

Metacognition is a higher-order thinking skill that is best described as thinking about thinking (Flavell, 1976). The most common method of measuring metacognitive skills in research is through a think- aloud technique in which individuals describe their thought processes verbally while completing a task or solving a problem. Clements and Nastasi (1999) used Sternberg's (1985) componential framework of cognition to qualitatively evaluate how the Logo programming language affected children's metacognition across a number of studies. They concluded that participating in Logo programming activities beneficially affected children's metacognitive thought processes, but emphasized the interaction of socio-developmental factors underlying metacognition on both a conscious and unconscious level. The authors call for a closer examination of

the evolution of metacognitive thought as it relates to programming in the Logo environment.

More recently, Allsop (2015) studied the way students in the UK thought throughout their programming experiences using a thought-mapping technique. The study included 30 ten to eleven-year-old children with one-hour prior programming experience in Scratch. Participants were first asked to diagram on a sheet of paper how they learn in any subject before they began a one hour per week computer game design course using the Alice programming environment. They were encouraged to update their “thinking maps” whenever they felt like it, and at the conclusion of the course, they were again asked to draw another thinking map. Results showed that children’s thinking maps became more continuous and “circular” during and after participation in the game design course, reflecting a trial-and-error, iterative approach to game creation and thinking. While methodological issues permeate the measurement of metacognitive thought processes, evidence seems to suggest that programming can positively affect the way in which children think about thinking.

#### *Academic Skills and Content*

Many of the early studies involving the relationship of academic skills to computer programming focused on mathematical thinking skills due to the intrinsic nature of the two. For example, Feurzig (1986) studied how Logo could be used to enhance student understanding of the algebraic concepts of variables and functions. Similarly, Noss (1986, 1987) examined how children learned algebraic and geometric concepts while programming in Logo. Olive (1991) analyzed text files of 30 ninth grade students’ work in Logo along three theoretical perspectives of understanding geometrical

relations and found that successful programming in Logo generally led to increased understanding of geometrical relations across the various taxonomic levels of the theoretical perspectives used in the analysis. Kafai (1995) investigated the ways in which fractions could be learned by deliberately incorporating them in student-designed games in the Logo environment.

More recently, Schanzer, Fisler, and Krishnamurthi (2013) developed the Bootstrap curriculum specifically to improve middle and high-school students' knowledge of algebraic concepts and coordinate geometry through guided game-making activities. Following the development of the Bootstrap curriculum, this group of researchers introduced the curriculum to over 500 students across states and cities in America from 2008-2012. In 2015, they looked at pre/post-test results of algebra problems for students who either completed the Bootstrap program ( $n = 123$ ) or were in a control group class ( $n = 26$ ) and compared their performance on word problems taken from the algebra section of the Massachusetts 8<sup>th</sup> grade standardized test in math. The programming syntax of the Bootstrap curriculum mimicked the manner in which mathematical functions were laid out in the selected assessment, e.g.,  $f(x) = x + 5$ , and activities connected the computer code to visual computer-based models representing mathematic principles, e.g., motion, variables, and geometry. Students who had training through the Bootstrap program performed significantly higher than students who did not partake in the training (Schanzer, Fisler, Krishnamurthi, & Felleisen, 2015).

While the Bootstrap curriculum seems to provide evidence that by directly incorporating or structuring lessons and activities around mathematic concept areas, mathematical ability is more dramatically improved, Columbian researchers Calao,

Moreno-León, Correa, and Robles (2015) analyzed students' math performance after programming in Scratch. They used a pre/post-test design incorporating an experimental and control group of 42 sixth grade students who either took part in Scratch programming activities over the span of three months, or attended their regularly scheduled math classes for the same time period. The researchers used a 16 item rating scale based on national standards put forth by the Ministry of Education of Columbia across four areas – modeling, reasoning, problem solving, and exercising –as their dependent variable. Students in the experimental group obtained significantly higher ratings of mathematical processing on all rated areas measured, showing large differences in the area of exercising, i.e., algorithmic thinking, while ratings of the control group declined across three out of the four areas. Although this study did not include any standard measure of student mathematical ability, and it was unclear the amount of computer programming time students received over the course of the three-month-long study, its design and inclusion of a comparison group that had general math instruction instead of Scratch programming provides further evidence that programming can aid in advancing student understanding and application of mathematical concepts.

Mathematics has been the primarily targeted academic skill area studied in relation to computer programming activities in education; however, there have been some studies investigating how other academic skill areas can be affected by incorporating computer programming activities in instruction (Baytak & Land, 2011; Hwang, Hung, & Chen, 2014). One such study conducted by Khalili, Sheridan, Williams, Clark, and Stegman, (2011) used the Game Maker software to create two and three-dimensional games during a summer program in an underserved American community. Sixteen high

school students spent two to three hours per day for four weeks in groups of four designing games based on the Federation of American Scientists' (FAS) "Immune Attack" educational science game to explain concepts in neurobiology with assistance from a lead classroom instructor knowledgeable in game programming, three college age mentors, and electronic communication with a scientist at the FAS. The researchers used interviews with students, classroom observations of the students at various points in the game design process, and email communication with the science subject matter expert from the FAS to evaluate student learning throughout the program. The authors analyzed these data and formed thematic conclusions about student learning indicating that students questioned their own knowledge about biology and voluntarily sought out answers to their questions, as well as demonstrated gains in their ability to explain and articulate complex scientific processes previously unknown to them. Although this study is qualitative in nature, and was composed of high school students, it highlights the deep level of understanding that can be achieved for specific academic content through programming activities rooted in game making.

In relatively rare study investigating how creating with computers could affect children's reading and writing skills, Owston, Wideman, Ronda, and Brown (2009) provided classroom instruction that incorporated the use of a game development program called Education Games Central in a large randomized control trial of 18 fourth grade classrooms. The researchers investigated whether creating games on computers could improve basic literacy skills when incorporated into regular classroom instruction across curricular units. Education Games Central draws upon the format of a variety of traditional board games, e.g., tic-tac-toe, and prompts users to generate questions related



to predetermined curricular objectives in order to move through the game. A standardized test of basic literacy skills with two forms (A and B) in a pre/post-test format, and an adapted version of a standardized test of written language as a post-test served as dependent variables. They found that students in experimental conditions performed significantly better than students in control groups on a subtest of the writing skills assessment measuring logical sentence construction that asked students to correct an illogical sentence; however, the effect was relatively small ( $\eta_2 = .031$ ). Qualitative observations and interviews with teachers showed that their perception of student learning in literacy improved more in experimental groups, and that teachers thought the skills learned in classes that used game development activities would be more likely to be retained by students. Although the programming activities students participated in throughout this study differ markedly from other literature in that students were not asked to construct computer code, but rather generate questions based on curricular content to be incorporated into a larger game program, it nonetheless provides some of the first quantitative evidence to suggest that writing skills can improve through participation in such activities.

### *Creative Thinking Skills*

The ability to think creatively is a crucial part of the human condition, and necessary to many aspects of effective problem solving and design processes. Measuring creativity, however, is inherently difficult as it involves degrees of situational spontaneity not often captured in a standardized way. One way to measure creativity that researchers around the world have used for many years is the Torrance Tests of Creative Thinking (TTCT) (Torrance, 1966). Rooted in Guilford (1967) and Torrance's (1969)

conceptualization of creativity, the TTCT measures divergent production characterized by fluency (the production of ideas), flexibility (the production of different ideational categories), originality (the production of unusual ideas), and elaboration (the persistency of introducing details to products) (Almeida, Prieto, Ferrando, Oliveira, & Ferrándiz, 2008). Almeida et al. (2008) used factor analysis to evaluate the construct validity of the TTCT in three large sample empirical studies from Spain and Portugal and found inconsistent factor structures that did not align with purported facets of divergent thinking claimed to be measured by the TTCT. Leandro, Lola, Mercedes, Emma, and Carmen (2008) have even posed questions to the validity of the TTCT's evaluation of creativity as a construct. Nonetheless, researchers have often turned to this measure as a valid way to measure creativity in research settings.

Clements (1986) looked specifically at how programming in Logo affected a variety of cognitive outcomes, and included the TTCT as a measure of creativity. Groups of first and third grade students worked in pairs twice per week across 22 weeks either on sequenced activities in Logo, or drill-and-practice educational software activities designed to teach academic content. The control group did not participate in computer lessons and attended class as usual. Results indicated that both first and third grade students in the Logo condition achieved significantly higher gains on the TTCT, specifically in the areas of originality and elaboration. The researcher notes, however, that the generalizability of the findings should be interpreted cautiously as students participating in the study had access to a dedicated and knowledgeable instructor throughout each training session. Clements (1986) does point out that the observed gains in cognitive outcomes could be attributed to the types of activities students engaged in

while using the computer rather than merely using the computer itself due to the inclusion of an experimental condition that involved computer instruction, but not programming.

A more recent research effort specifically to focus on creative problem solving as it relates to computer programming was conducted by South Korean researchers Kim, Chung, and Yu (2013), who used a sequence of training modules in the Scratch programming environment designed to promote real-world creative problem solving skills through explicit instruction to solve programming problems within a six-step problem solving framework designed to promote imagination, creation, collaboration, and reflection by using various commands to program coded sequences. Both typically developing adolescents in South Korea ( $n = 119$ ), and those identified as gifted ( $n = 30$ ), were randomly assigned to treatment and control groups followed by 16 weeks of training sessions in Scratch. The researchers developed a synthetic creative problem solving test (SCPST) based on previous work by the Korean Educational Development Institute (ED.) (Cho, Jang, Jung, Lim, & Park, 2002) to use as their primary measure of student creativity. The test includes word problems that prompt students to develop creative solutions to problems, e.g.,

Now the world is making an effort to protect depleting energy resources. Bicycles are especially popular because they do not consume energy resources and emit exhaust. What would you do if you could improve the disadvantages of the current bike? Let's write as much as possible about an idea to upgrade anything that is inconvenient. (Kim, Chung, & Yu, 2013, pp. 177)

They validated the test by pre-testing a separate group of students with the measure and comparing correlations of test scores to student scores on other instruments claiming to measure creativity, e.g., the TTCT. The test correlated moderately with the TTCT Figural test ( $r = .62$ ), but there was very little correlation with the TTCT Verbal test ( $r = -.48$ ).

The authors explained this low correlation with the TTCT Verbal by arguing that because the content of the questions was science-related, only logical answers received a score; thus impacting the way scores are calculated and compared across measures. The authors thus concluded that the SCPST was a valid measure of divergent thinking, logical thinking, and scientific problem solving ability and scored student responses the following five dimensions: fluency, elaboration, sensitivity, openness, and flexibility. Results showed significant increases in digital fluency, which was defined by the researchers as the ability to solve real life problems creatively using digital technology, and originality. The researchers concluded that computer programming training can be used to solve realistic problems and can enhance creative problem solving skills that prove useful in many facets of life.

#### Assessing Learning in Computational Thinking

While there appears to be evidence that computer programming can improve various cognitive and academic skills, the assessment of CT skills themselves poses significant challenges, and is an area in which ongoing development in the field is taking place. A variety of procedures and assessment techniques have been used in the literature to assess student learning and development of CT skills. These range from think-aloud interviews with students about their experiences and projects (Hwang, Hung, & Chen, 2014; Khalili, Sheridan, Williams, Clark, & Stegman, 2011), to analyses of the frequency of types of code incorporated into projects (Baytak & Land, 2011; Denner, Werner, & Ortiz, 2012; Werner, Denner, & Campe, 2014), interactive debugging tasks (Werner, Denner, Campe, & Kawamoto, 2012; Su et al., 2014), and multiple choice assessments (Grover, Pea, & Cooper, 2015; Straw, Bamford, & Styles, 2017). Some studies use

qualitative descriptions or case studies based on observations to describe how and what students learn during the course of digital media creation, with a process-oriented, descriptive, and exploratory approach (Fessakis, Gouli, & Mavroudi , 2013; Kafai, Peppler, & Chiu, 2007; Maloney, Peppler, Kafai, Resnick, & Rusk, 2008). Each assessment method has its unique pros and cons, selected according to the focus of the investigation. What remains clear, however, is that a valid and reliable CT assessment remains to be developed.

Adding to the growing need to define how to evaluate CT skills, Brennan and Resnick (2012) not only outline a theoretical model to understand how to conceptualize computational thinking, but also summarize three ways in which researchers and educators can assess student learning, specifically in Scratch, and the associated pros and cons for each. The first approach discussed utilizes a visual analysis of Scratch projects or portfolios of using a tool called a Scrape visualization developed by researchers at the College of New Jersey (Wolz, Hallberg, & Taylor, 2011). A Scrape visualization displays blocks of code used by a Scratch user arranged in rows and color coded according to the type of command block, with columns representing individual Scratch projects. This type of analysis allows researchers to quickly assess an individual's usage of various types of code, as well as types of code that were not utilized at all. Furthermore, researchers are able to inspect the evolution of a Scratch user's use of code over time. In one example of this type of assessment technique, Chang, Tsai, and Chin (2017) recently developed the Dr. Scratch "web crawler" tool intended for analyzing Scratch users' projects according to seven CT principles (flow control, data representation, abstraction, user interactivity, synchronization, parallelism, and logic), assigning a score for each CT skill area, and

providing data on student usage of various types of code. This type of web-based analytic tool produces data about Scratch users' projects quickly and efficiently, and could be useful for instructors and researchers alike.

As with all product-oriented analyses of computational thinking, one limitation to these types of analysis is that usage of code does not necessarily denote understanding of the code. The second approach discussed by Brennan and Resnick (2012) involves artifact-based interviews to evaluate individuals' conceptual understanding of how and why their projects functioned. This type of analysis is more labor intensive, and thus less conducive to large-scale research efforts; however, it allows researchers to more fully understand whether individuals truly grasped the computational concepts that enabled their projects to function. A third approach, design scenarios, presents users with pre-designed Scratch projects and asks them to explain what the project does, describe how it could be altered and expanded, correct any mistakes, and modify the project by adding a new element. This approach to assessing CT skills allows researchers to systematically study how computational concepts and practices change over time, and enables users to demonstrate their knowledge in the moment rather than recalling and verbalizing an explanation at a later time. Variations of these three lines of assessing CT presented by Brennan and Resnick (2012) have been used across studies not only in with Scratch, but also in other novice-oriented programming environments.

One example of an interactive task-based assessment of CT skills is the "Fairy Assessment" developed by Werner, Denner, Campe, and Kawamoto (2012) for the Alice programming environment. The Fairy Assessment was designed to measure two computational thinking principles identified by the Carnegie Mellon Center for

Computational Thinking, i.e., thinking algorithmically, and making effective use of abstraction and modeling. The assessment uses the Alice programming environment, which is an environment especially conducive to story-based games, to independently assess whether students could execute programming tasks that were designed to represent knowledge of the two CT skills under examination. Successful completion of the task presumably relates to the underlying CT skills embedded within the task, suggesting that this type of assessment is more thorough than simply documenting the types of code contained within students' projects, and the methodology can be applied similarly to other programming environments.

To further understand whether students actually learned CT skills and could demonstrate their application in a novel environment, i.e., transfer, Grover, Pea, and Cooper (2015) measured how transfer of CT skills from the Scratch programming environment to a text-based programming environment, e.g., samples of Pascal/Java-like code borrowed from past AP exams in computer science. The preparation for future learning (PFL) exam was administered to groups of students who had participated in a structured Scratch coding class to see if their knowledge of Scratch could be applied to a novel programming environment. The researchers explained and provided the syntax of the new programming language before administering the test, which occurred at the end of a seven-week Scratch curriculum. Results indicated that students were able to apply some computational concepts, but struggled with applying concepts of loops and variables in this new environment. The authors conclude that relatively weak student performance on the PFL test, which included a large portion of items that required understanding of loops and variables (concepts that students already had difficulty with in

the Scratch environment), focused too much on concepts that were not explicitly taught in the Scratch curriculum selected for the study. The results of this study may be somewhat disheartening for researchers hoping to demonstrate that engagement in novice-oriented visually based programming languages like Scratch could be applied in more prevalent text-based programming languages; however, students were able to transfer many ideas learned in the Scratch curriculum to an environment in which they had extremely limited experience.

An innovative analytic technique has recently been developed to understand on a micro level how students use and modify their Scratch projects (Fields, Quirke, Amely, & Maughan, 2016; Fields, Quirke, Horton, Maughan, Velasquez, Amely, & Pantic, 2016; Pantic, Fields, & Quirke, 2016). The technique utilizes large amounts of JSON files, which are text-based versions of Scratch projects, collected through the backend of the Scratch programming environment in combination with front-end Scratch projects, observations, and interviews to understand how students used various code blocks during their work in the Scratch environment. The technique involves capturing snapshots of the code blocks being used by students by saving JSON files every two minutes, or when students switch from editing costumes, backgrounds, or sound back into coding. The researchers developed a parser to analyze how students used various categories of code within and across training sessions, and combined these data with interviews and observations to better understand how students used code and understood computational concepts. The results of the researchers' efforts to more fully comprehend the ways in which students use computational practices, e.g., remixing and debugging, and grasp computational concepts shows promise and will undoubtedly help researchers and



computer science educators alike to develop new strategies for developing more effective lessons, and evaluate student learning.

Finally, in a large-scale randomized control trial of coding clubs with 317 students in 21 schools in the UK, Straw, Bamford, and Styles (2017), working with the Raspberry Pi foundation, utilized a unique measure of CT skills called the Bebras Challenge, which is an online timed, 15-item set of scenarios related to CT areas as conceptualized by Selby and Woollard's (2013), to measure whether nine to ten-year-old students learned CT skills after participating in a year-long coding club that was composed of a mixture of Scratch, HTML/CSS, and Python activities and lessons. The Bebras Challenge tasks involve no prior knowledge of programming language, so they were suitable for a post-test in the control group. Items consist of multiple choice responses, and students can sometimes interact with item response choices by trial-and-error, which can be akin to testing or debugging. For example, students can test out whether their sequences of directional arrows will get an object through a maze. This set of CT-oriented tasks is part of an ongoing international competition originating in Lithuania in 2004 to better understand students' ability to think computationally, and promote the field of CS (Román-González, Pérez-González, & Jiménez-Fernández, 2017). In 2015, 1.3 million students from 38 countries in the 2015 challenge (Izu, Mirolo, Settle, Mannila, & Stupurienė, 2017). Students who participated in code clubs in the Straw, Bamford, and Styles (2017) study did not show a measurable improvement in computational thinking as measured by the Bebras Challenge CT assessment when compared to control students who did not attend code clubs, but they did show significant improvements in their skills within Scratch, HTML/CSS, and Python.

## State of the Field

From the above review of the literature, it is clear that participation in computer programming activities in a variety of programming environments and pedagogical strategies can impact learning not only in the domain of computational thinking, but also in higher-level cognitive abilities (problem solving and metacognition), specific academic content, (science, math, and literacy), and creative thinking abilities. The ways in which computers have been used in the classroom have undergone significant changes over the past 30 years, and a return to the fundamentals of computer programming as an important 21<sup>st</sup> century skill is well underway across primary and secondary classrooms around the world. While researchers in the 1980s and 1990s studied thought processes associated with computer programming in educational environments, and found some evidence for learning gains, their results lacked an underlying conceptual framework to synthesize and make sense of what children learned and the manner in which they learned. The relatively recent emergence of the term computational thinking is a way to operationalize the ways in which children learn computer science concepts and practices. Translating the findings of early computer science education researchers within the CT paradigm has been suggested as an important endeavor to inform current research and practice (Grover & Pea, 2013).

The assessment of CT is also an area that continues to develop as researchers further refine the definition of what it means to think computationally, and create new ways to evaluate what and how students learn while they code. Despite some evidence that highly valued cognitive skills and academic knowledge can be improved through computer programming activities, questions still remain regarding how educators can

best support the development of computational thinking in the school environment. The focus on improving traditionally valued academic competencies serve as an obstacle for educators, policymakers, and researchers to overcome in order to successfully integrate coding in the classroom. Movements to create clearly outlined standards across age ranges and content areas for the field of computer science, and train teachers to integrate computational thinking activities and computer science lessons to teach specific curricular content is underway; however, several barriers exist that have slowed the integration of computer science into general education. These include the perceived difficulty of computer science principles by general education teachers, lack of school resources (space, finances), and administrative focus on improving core academic performance. Occupational competencies related to computer science and information technology skills are in high demand, and therefore, it is more important than ever to provide high quality empirical evidence to understand whether providing students with the opportunity to develop computational thinking competencies may impact other areas of their academic performance, so that administrators and teachers alike may be more likely to adopt CT activities into their curricular content.

## CHAPTER 3

### METHODS

The current study aimed to shed light on the relationship between computer programming activities, problem solving ability, academic achievement, and creative thinking in ten to fourteen-year-old children participating over summer educational programming at a public charter school in an urban school in the mid-Atlantic region of the United States. Youths participated in Scratch-based computer programming activities following a semi-structured set of lessons, led by an instructor with experience in education and computer science. The first two days of the study consisted of individual and group assessments for all participants in the study. Then, two sequential classes of participants (henceforth referred to as experimental and control groups) were provided instruction, with a second round of assessments occurring after the experimental group ended Scratch lessons; this portion of the study constituted the controlled trial phase. Finally, a third round of assessments occurred after the control group had completed Scratch lessons.

#### Participants

The partnering school was selected to participate in the study primarily because the school offered a summer-long day-camp program open to all students, but secondarily because the student demographic represents a traditionally underserved population. Youth attending the summer programming were provided with an explanation of the study's procedures, and the potential risks and benefits to their participation individually by the primary student investigator. Families of youth who assented to participating in the study then met individually with me to go into more detail about the study's details over

the course of a routine summer enrollment school meeting. Parents who were unavailable to meet in person were sent home consent forms with contact information for myself and the principal investigator. There were two youth who did not assent to participate, and two who began the study but changed their mind about participating after beginning instruction for one or two days. Upon completion of the study, all participant assessments, as well as any other identifying documents collected throughout the study, were de-identified and participants were assigned a random two-digit number.

In total, there were 24 youth who participated in any aspect of the study; however, some youth's attendance was such that they only participated in the initial assessment portion of the study ( $n = 6$ ), or for less than or equal to half of the instructional time ( $n = 5$ ). Five boys and seven girls comprised the experimental group, while eight boys and four girls comprised the control group. Youth ranged from ten to fourteen years-old, with an average age of 11.5 years-old (11.63 in the experimental group, and 11.38 in the control group). All participants had just completed their respective grades, and there were five fourth graders, two fifth graders, four sixth graders, and one seventh grader in the experimental group; while there were four fourth graders, five fifth graders, two sixth graders, and one seventh grader in the control group. There were four participants who received special education services in the area of learning support, with two of these youth in both the experimental and control groups.

The lead computer science educator was recruited through email postings across a variety of listservs related to computer science education. She received a Bachelor's of Business Administration in Management Information Systems and Accounting from a State University in 2012, and completed a Full Stack Web Developer Online Program in

2017, prior to the study's start date. She had extensive field experience working in healthcare system implementation, developing training materials and lesson facilitation, and building basic websites and apps. In addition, she had previously used the Scratch programming environment while serving in a volunteer capacity with middle school students for Girls Who Code, and Tech Girlz –two non-profit organizations dedicated to STEM education for young women in the area. Prior to the study's start date, I met with her to discuss her experiences and interest in the study, and after she agreed to participate in the study, provided her with the tentative lesson plan and schedule for curriculum selected for the study. She independently reviewed and completed the lessons to familiarize herself with all the nuances and details, and participated in a follow-up troubleshooting meeting to resolve any anticipated difficulties before classes began. She acted as lead instructor throughout the duration of the study, while I supported her with behavior management in the classroom, and lesson planning. The partnering school also officially brought her on as a summer programming educator to provide financial reimbursement for her time.

Graduate research assistants studying the field of School Psychology were recruited via electronic postings at Temple University in the winter and spring of 2017 to conduct individually-administered standardized assessments of problem solving abilities and math achievement with participating youth. A total of five research assistants worked as field assessors throughout the duration of the study. Group training sessions were held in the spring of 2017 to introduce assistants to each of the selected measures of the study, and also to detail alterations to standardized administration for one of the assessments. Follow-up observed testing sessions were held with each research assistant, with myself

as a mock examinee making both common and unusual responses in order to ensure assistants were prepared for the diversity of potential participant responses. A standard checklist provided by the publisher of the Woodcock Johnson Tests of Cognitive Abilities, Fourth Edition, was used to ensure proper administration of this assessment. Assessors were provided with a physical and digital copy of changes to standardized administration of this assessment to further reduce the likelihood of administration error. Each assistant reached administrative proficiency for reliable and valid administration for the assessments selected for the study.

Another group of three research assistants (two graduate students and one undergraduate student) were recruited in the winter of 2018 to assist with assessing creative thinking and creative problem solving through electronic postings on various listservs for graduate students in the College of Education at Temple University and the Computer Science Teachers Association, Philadelphia Chapter. The responding research assistants shared a commonality in that they had prerequisite interest and knowledge of creativity research and cognitive assessment in a collegiate setting, thus satisfying criteria to be considered quasi-experts (Kaufman & Baer, 2012). One of the research assistants was an advanced doctoral candidate at the College of Education at Temple University and a high school CS teacher. Another assistant was a Masters level graduate student with knowledge and experience in the field of assessment of learning, also from the College of Education at Temple University. The undergraduate research assistant was a senior studying Psychology at a nearby university who was recommended as a qualified candidate to act as a research assistant by his neuropsychology course instructor.

The recruitment email message contained brief details about the study and the role of the assistants, in addition to a digital copy of Amabile's (1982) article describing the theoretical framework and process of the assessment method. A conference call was subsequently held to provide an overview of the study, review the assessment method and procedures, and ensure each research assistant was provided with uniform training. Research assistants (raters) received and returned their assessment packets in-person or through the mail, with individual participant responses arranged in a random order but grouped according to the assessments corresponding time point, i.e., pre/post. Two of the research assistants provided ratings on the standard, sequential presentation of the items (1, 2, 3, 4), while one of the research assistants provided ratings on an altered presentation of items (3, 4, 1, 2). This decision was made to control for any order effects. Packets also included pre-populated rating forms, and an additional hard copy of the assessment procedures. Assistants were instructed not to communicate with one another about their ratings.

### Design

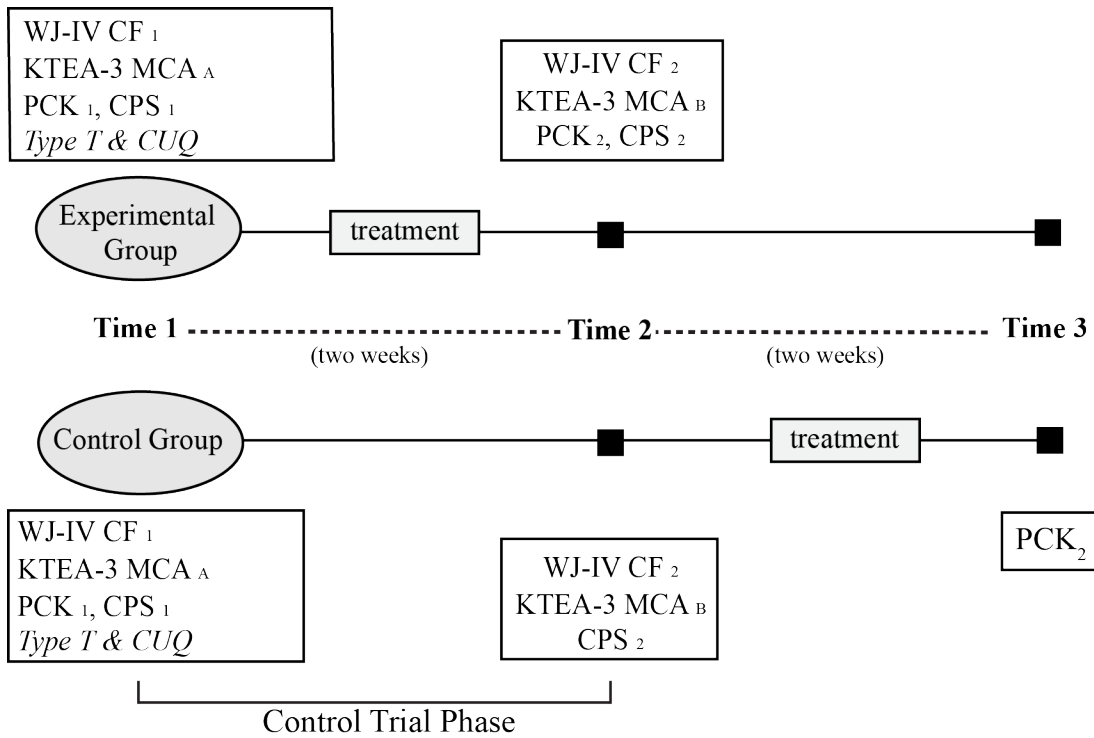
Due to the scheduling conflicts that would have arisen by randomizing children enrolled in the participating school's program purely for study purposes, participants were divided into an experimental and control group to accommodate previously planned camp activities and trips. A total of 12 children were assigned to both the experimental and control groups; however, some children in the control group either chose not to continue their participation in the computer class ( $n = 2$ ), or did not attend any of the instructional days for unknown reasons ( $n = 3$ ). Assessment data for these youth were, therefore, incomplete. Youth in the experimental group completed the computer coding



class first, while those in the control group attended regularly scheduled daily activities along with other summer camp children not enrolled in the study. Youth in the control group then completed the computer coding class while those in the experimental group attended regularly scheduled daily activities along with other summer camp attendees not enrolled in the study. The decision to incorporate both experimental and control groups in the computer class over the course of the study was made in order to comply with ethical principles, and general best practice of equity in research; moreover, the potential educational benefit of participation in the computer course was the same across both experimental and control groups throughout the course of the study. The planned delayed treatment control trial experimental design was intended to allow for strong conclusions to be made regarding the efficacy of the computer class on cognitive and academic variables (Chambless & Hollon, 1998). The study also incorporated a pretest-posttest design element such that children in both the experimental and control groups completed all measures before beginning the computer class and after completing the class.

Figure 3.1 graphically depicts the design of the study with the associated measures that were administered along the various time points in the study. In order to preserve the psychometric properties of the standardized measures selected for the study (WJ-IV CF and KTEA-3 MCA), it was only possible to administer these measures at two time points; consequently, the control trial component of the study took place between time points one (T1) and two (T2), and also included the CPS assessments. In order to measure changes in participant knowledge of computer programming concepts, each group was assessed for programming conceptual knowledge prior to beginning the class (T1), and again upon completion (T2 for experimental group and T3 for control group).

The decision to administer both the CPS and PCK group assessments to youth in both groups at the start date of the study was made to increase the internal validity of the study by preventing the transmission of assessment content between children in experimental and control conditions during regularly scheduled summer programming. The assessment of programming conceptual knowledge, thus, represents a more traditional pretest-posttest experimental design, specifically measuring participant changes in programming conceptual knowledge after participating in computer programming activities, but not compared to a control group.



*Note:* WJ-IV CF<sub>1</sub> & 2: Woodcock Johnson Tests of Cognitive Abilities, Fourth Edition - Concept Formation, pre and post-test; KTEA<sub>A</sub> & B: Kaufman Tests of Educational Achievement, Third Edition - Mathematical Concepts and Applications, Forms A and B; PCK<sub>1</sub> & 2: Assessment of Programming Conceptual Knowledge, pre and post-test; CPS<sub>1</sub> & 2: Assessment of Creative Problem Solving, Pre and Post-test; Type T: Type-T Personality Questionnaire; CUQ: Computer Usage Questionnaire.

Figure 3.1. Experimental Design and Measures

### Measures

A variety of measures were used to assess problem solving ability, academic achievement, programming conceptual knowledge, prior programming experience and computer usage, and creative problem solving. In addition to direct individual and group assessments, the partnering school provided each child's most recent Pennsylvania System of School Assessment (PSSA) scores in English and Language arts, and Math. Scaled PSSA scores for each area were collected and used in data analysis.

To gauge participant levels of problem solving ability, an adapted version of the Concept Formation subtest of the Woodcock-Johnson Tests of Cognitive Abilities, Fourth Edition (WJ-IV) (Schrank, McGrew, & Mather, 2014) was developed to accommodate a two time-point administration that did not consist of identical items. The WJ-IV Concept Formation test is a measure of inductive reasoning, i.e., the ability to observe a phenomenon and discover the underlying principles or rules that determine its behaviors (Flanagan, Ortiz, & Alfonso, 2013). In the WJ-IV Concept Formation subtest, examinees are presented with a complete stimulus set, and he or she must derive the rule for each item; therefore, in addition to measuring principles of inductive logic, the WJ-IV Concept Formation subtest also measures the mental flexibility required when an individual shifts mental set, which is an aspect of executive processing (Mather & Wendling, 2014). An expert in the field of cognitive assessment was consulted to split the subtest items into two separate, and psychometrically similar measures, thereby reducing the chance of practice effects associated with completing identical measures within the short test-retest time interval. Each participant completed all sample items on each testing occasion to maintain the integrity of the test; however, the test at time point one was composed of half of the item pairs appearing on a single page of the stimulus book, while the test at time point two was composed of the remaining half of item pairs.

The intuitive link between applied mathematical problem solving and computer programming activities justified incorporating an assessment of mathematics into the current study. Although lessons within the selected curriculum did not explicitly incorporate math instruction into daily activities, in order to create functional computer programs, knowledge of math concepts and their applications was necessary; therefore,

the Math Concepts and Applications (MCA) subtest from the Kaufman Tests of Educational Achievement, Third Edition (KTEA-3) (Kaufman & Kaufman, 2014) was selected as a way to gauge changes in mathematical thinking skills. Not only is this test a standardized measure of mathematical conceptual knowledge, but the KTEA-3 also offers two forms (A and B) to allow for subtests to be administered in close temporal proximity to one another while maintaining technical adequacy and eliminating practice effects associated with completing identical assessments in close test-retest intervals. Form A was administered to all participants in attendance before the study began, and form B was administered to all participants in attendance at time point two. The KTEA-3: MCA subtest has a split-half internal consistency reliability coefficient of .96 for fifth grade students, and is highly correlated with ( $r = .85$ ) with the Mathematical Problem Solving subtest of the Wechsler Individual Achievement Test, Third Edition (WIAT-III); therefore, it was considered a valid and reliable measure of academic achievement in the domain of applied mathematical problem solving.

To measure creative thinking abilities, the consensual assessment technique (CAT) (Amabile, 1982), using three raters trained on the method and considered to be quasi-experts in the field of learning, computer science, or educational psychology, was utilized. The CAT method of assessing creativity is considered by leading creativity researchers to be the most reliable and valid manner in which creativity can be measured (Baer & McKool, 2009). The CAT involves rating participant-created products and artifacts using a scale across the two major components considered essential in evaluating creativity, i.e., originality and usefulness (Mayer, 1999). An assessment of creative problem solving (CPS) was developed using items that were similar to a study conducted

by South Korean researchers, Kim, Chung, and Yu (2013) who developed and validated the “Synthetic Creative Problem Solving Test” using the CAT to evaluate dimensions of creativity with over 100 youth who participated in Scratch programming activities. The primary measure of creative problem solving in the current study consisted of a pre and post-test with four prompts nearly identical to those used in Kim, Chung, and Yu’s (2013) study; however, the language of four item prompts composing each test was modified to reflect cultural and geographical differences between South Korean and American students (see Appendices A and B for complete pre and post-tests).

Two brief questionnaires were completed by all participants in the study to investigate the relationship between personality differences, and previous programming experience or computer usage, and assessment variables. Participants completed a brief, eight-item questionnaire designed to measure the degree to which they endorsed taking risks and seeking out new or exciting experiences, referred to as the Type-T (Thrill) dimension of personality (Farley, 1986). A recent review of the Type T literature is found in Sarshar (2017), which explored the personality characteristic’s relationship to mindset, flourishing, psychological entitlement, creativity, and stress in a sample of undergraduates. The questionnaire presented statements like, “I enjoy taking chances,” and, “I like to make up my own mind.” Participants then circled, “Never,” “A little bit,” or, “A lot” (Farley, 2017, personal communication). See Appendix C for the full Type-T questionnaire. The Computer Usage Questionnaire (CUQ) asked participants about their prior participation in computer programming activities, and their usage of computing devices both at home and in school. Both questionnaires were completed in small groups at the beginning of the study. These data were used to provide information on how

children with different personality characteristics or varying levels of experience using computers and using code may or may not be more likely to benefit from computer programming activities.

Finally, youth in both the experimental and control groups were given a test of computer programming conceptual knowledge (PCK) at the beginning of the study (PCK<sub>1</sub>), and again after each group had completed the computer coding course (PCK<sub>2</sub>). For the PCK<sub>1</sub>, participants completed the most recently available 2016 USA Bebras Challenge Computational Thinking Assessment, obtained through coordination with the USA Branch of the Bebras Organization (“Bebras Computing Challenge,” 2018). The pre-test (PCK<sub>1</sub>) was selected for its attempt to assess CT skills with tasks that do not require any prior knowledge of computer programming, combined with its use in prior research (Straw, Bamford, & Styles, 2017). See Dagienė and Stupurienė (2016) for a review of studies involving the Bebras Challenge. Items on the PCK<sub>1</sub> were adapted from the 2016 UK Bebras Challenge Computational Thinking Assessment to reflect language differences between UK English and American English, while retaining the same content and tasks (“UK Bebras Computational Thinking Challenge,” 2016). See Appendix D for questions and answers for the item set of the USA 2016 Bebras Challenge used in the current study, and for a description of the theoretical and practical considerations in the early development of the Bebras contest, see Dagienė (2006).

The assessment is an online, 45-minute limit, 15-question test assessing various areas of CT as conceptualized and described by Selby, Dorling, and Woollard (2014), i.e., algorithmic thinking, evaluation, decomposition, and generalization, and each item requires tapping in to up to three of these CT areas. Each item was developed through a

workgroup of CS researchers and educators that meets on an annual basis, and is presented in a multiple-choice format that presents a child-friendly, illustrated scenario designed to 1) represent key computational concepts, 2) be easily understandable, 3) solved within a three-minute timeframe, 4) able to be presented on a single page, 5) solvable at a computer without the use of other software or paper and pencil, 6) independent from specific systems, and 7) be interesting and/or funny (“Bebras,” 2018). Item sets are grouped for specific age-bands (Pre-Primary – ages 5-8, Primary – ages 8-10, Benjamins – ages 11-12, Cadets – ages 13-14, Juniors – ages 15-16, and Seniors – ages 17-18), and each item set contains five items across three levels of difficulty. Items within each age band are adjusted annually to contain specific tasks that reflect the expected range of performance across age groups based on observed patterns of performance from the previous year. Participants in the current study, regardless of their chronological age, were administered the Benjamins group item set.

Participants were assigned anonymous “skeleton” accounts to access assessment content through the web, and were presented items in a random order until the time expired, or all questions had been answered. Participants were allowed to ask questions while they took the assessment, and both myself and the lead course instructor provided task clarification when needed. Attempts were made to limit communication among nearby children; however, the configuration of the classroom inevitably allowed for some peer communication during the assessment. See Table 3.1 for a list of the items administered to participants in the current study, and their associated CS domains, CT areas, and key word tags describing specific concepts embedded in the task. The table is organized according to item difficulty level.



Table 3.1

2016 USA Bebras Challenge (PCK<sub>1</sub>) Assessment Composition

Item Name	Difficulty	CT Skills / <u>CS Domain</u> / (keyword tags)
Mazes	A	Algorithmic Thinking / <u>AP</u>
Soccer Game	A	Algorithmic Thinking, Evaluation / <u>AP</u> / (IF condition)
Bottles	A	Abstraction, Evaluation / <u>DSR</u>
Tube System	A	Algorithmic Thinking, Decomposition, Generalization / <u>AP</u>
Party Guests	A	Algorithmic Thinking, Decomposition / <u>AP</u> / (Dependency, Graph)
Secret Recipe	B	Algorithmic Thinking, Decomposition / <u>DSR</u>
Car Trip	B	Algorithmic Thinking, Decomposition / <u>AP</u>
Robot Exit	B	Algorithmic Thinking / <u>AP</u>
Party Banner	B	Abstraction, Evaluation, Generalization / <u>AP</u>
Beaver Code	B	Algorithmic Thinking, Decomposition, Generalization / <u>DSR</u>
Blossom	C	Evaluation, Generalization / <u>AP</u>
Magic Potions	C	Algorithmic Thinking, Evaluation / <u>AP</u>
Hurlers Shake Hands	C	Algorithmic Thinking / <u>CPH</u> / (Parallel processing)
Primary Health Care	C	Abstraction, Evaluation / <u>Data</u>
Paint it Black	C	Abstraction, Algorithmic Thinking, Evaluation / <u>AP</u> / (Boolean Algebra)

Note: In CS Domain column, AP = Algorithms and Programming; DSR = Data, Data Structures and Representations; CPH = Computer Processes and Hardware.

A post-test of programming conceptual knowledge (PCK<sub>2</sub>) was developed largely based on the items from the Coding Quiz used in Straw, Bamford, and Styles (2017), and through collaboration with the lead course instructor. It was administered as a multiple-choice, paper-pencil assessment consisting of seven items that focused on the seven core computational concepts as outlined by Brennan and Resnick (2012), i.e., sequences, loops, events, parallelism, conditionals, operators, and data (see Appendix E for the complete PCK<sub>2</sub> assessment). This assessment was selected and developed due in part to its practical feasibility with respect to the total amount of time children spent completing various assessments, while also in part due to the efficiency with which responses could be quantified. Participants were presented with a prompt describing a sample Scratch project with six, lettered choices containing different samples of Scratch code, and one, “I don’t know,” option. Item content also reflected specific instructional activities and language taught and used during the course, and in this way served to function as an assessment of the degree to which participants learned course content. Table 3.2 summarizes the item task, CT areas, and curricular content for the PCK<sub>2</sub>.

This type of assessment technique for knowledge of programming concepts has notable advantages to the more prevalent artifact-based assessment technique (e.g., Denner, Werner, & Ortiz, 2012). First, knowledge of computational concepts is directly tested as opposed to indirectly assumed through analysis of artifacts, e.g., percentages of types of code included in projects; and second, this assessment technique focuses on a process-in-action rather than a process-via-memory inherent in interview, or self-report assessment techniques (Brennan & Resnick, 2012; Werner, Denner, & Campe, 2014).

Table 3.2

*Programming Conceptual Knowledge Post-test (PCK<sub>2</sub>) Composition*

<b>Item</b>	<b>Task Description</b>	<b>CT Areas</b>	<b>Curricular Content</b>
1	Move a cat along a path to a donut.	Sequences	Understanding of rotation, direction, and orientation
2	Make a teacher say something by inputting the correct value.	Conditionals Operators Data	Understanding of > symbol, "ask" and "say" blocks
3	Make a windmill rotate forever.	Loops Sequences	"Forever" block
4	Move a parrot when a key is pressed.	Events	Understanding of x and y as axes
5	Make a dinosaur say a times table.	Sequences Loops Operators Data	Understanding of > and * symbols, Variables
6	Make a person dance and speak forever.	Parallelism Sequences Loops Events	"Define," "Broadcast," and "Receive" blocks
7	Set a timer to a song.	Events Sequences Loops Operators	Timer as a variable, "Repeat Until" block

Participants in both experimental and control groups spent the first two days of the school's summer camp educational program completing individual and group assessments, and setting up online accounts necessary for using the selected curriculum and programming environment. Upon completion of lessons and activities of the selected curriculum, participants then completed individual and group post-tests associated with

their respective group condition and time point. Individually administered assessments (i.e., the WJ-IV CF<sub>1 & 2</sub>, and the KTEA-3 MCA<sub>A & B</sub>) took approximately 15 minutes to administer per child. All other assessments were administered in small groups with the PCK<sub>1</sub> taking on average approximately 30 minutes for participants to complete; the PCK<sub>2</sub> approximately 15 minutes; and the CPS<sub>1 & 2</sub> assessments taking approximately 15 minutes each for participants to complete. The Type T personality questionnaire, and the Computer Usage Questionnaire took approximately 15 minutes per participant to complete altogether. The approximate amount of time for all assessments included in the study was approximately 150 minutes per child.

### Materials

The curriculum selected was developed by the ScratchEd team at the Harvard Graduate School of Education, and is entitled “Creative Computing” (Brennan, Balch, & Chung, 2014). The Creative Computing (CC) curriculum is composed of seven units with 44 lessons and activities range from 19.5 to 30.25 hours of time in total. The sequence of lessons and units introduces users to the Scratch programming environment and explores the various functions and tools that make up Scratch. Lessons include non-computerized exercises designed to encourage children to think about the intricacies of designing a computer program to perform a desired operation or function, as well as open-ended project-based activities, e.g., personalized story-based animations, individually created games, etc., that allow children to personalize their experience in the Scratch programming environment. Participants were provided with individual, hard copies of workbooks to work through lessons, take notes, and write their ideas and reflections instead of using individual design journals, as is suggested in the CC curriculum. Brief

group discussions based on the prior day's activities, occurring at the beginning of each instructional day allowed for peer feedback and collaboration, which has been previously linked to greater understanding of programming concepts (Werner, Denner, & Campe, 2014). When youth finished lessons early, they were encouraged to complete a series of mini-lessons called "Scratch Cards" developed by the ScratchEd team. These Scratch Cards served as mini-lessons designed to teach specific functions of various codes across the various code categories, and were grouped as themed sets, e.g., Animate Your Name, Create a Story, Make Music, etc.

Each lesson was introduced in a group setting, where approaches to activities embedded within each lesson were discussed together, and then demonstrated by the instructor on a projector. Youth suggested codes to input, while the instructor talked through how the computer interpreted the code. When the proposed solution did not produce the desired outcome, the instructor used a think-aloud procedure to model the problem solving process until the outcome was achieved. Youth then individually worked to create Scratch projects while I and the lead instructor circulated through the room to work individually with participants to help troubleshoot malfunctions in their projects, and also to help them understand how to use various codes, upload media, and navigate the Scratch interface. The lead computer instructor and I met briefly after each class period to discuss our observations and experiences as it related to how the youth were responding to the instructional activities. We then communicated electronically to adjust the following day's lesson plans by developing sample projects illustrating key concepts using the interests expressed in reflection discussions in an attempt to boost interest and

motivation. A complete list of lessons completed for each group across instructional days is displayed below in Table 3.3.

Table 3.3

*Day-by-day Lessons for Experimental and Control Groups*

<b>Experimental Group</b>		<b>Control Group</b>	
Day 1	Introducing Scratch Scratch Account Scratch Surprise	Day 1	Introducing Scratch Scratch Account Scratch Surprise
Day 2	Programmed to Dance 10 Blocks	Day 2	Programmed to Dance Debug It!
Day 3	Debug It! About Me Intro to Build-a-Band	Day 3	10 Blocks About Me
Day 4	Orange Square, Purple Circle Performing Scripts Build-a-Band	Day 4	Performing Scripts Build-a-Band
Day 5	It's Alive! Debug It!	Day 5	Music Video
Day 6	Music Video	Day 6	Characters Conversations Scenes
Day 7	Characters Conversations	Day 7	Debug It! Dream Game List Starter Games
Day 8	Debug It! Scenes	Day 8	Score Extensions Interactions
		Day 9	Debug It! Extensions/Remixing
		Day 10	Robotic Kit Activity

The total duration of each class was an attempt to reflect international guidelines on the amount of CS instruction required for a high school level certification in computer science and information technology instruction set forth by the Department of Education in the UK, and also adopted by South Korea (Yoo et al., 2006). Each class session lasted

approximately three hours per day. The experimental group participated in Scratch activities over eight class periods while the control group participated in Scratch activities for ten class periods in total, amounting to 24 hours of experience with Scratch for the experimental group, and 30 hours of experience with Scratch for the control group. This difference in the number of experiential hours was a result of losing instructional days due to excessive heat school closures, and a scheduling miscommunication.

A positive reinforcement system using salvaged silicon microchips as tokens was established prior to the start date of the study in a preemptive attempt to promote positive behavior among youth in each group. Participants could earn a chip for 1) helping another classmate solve a problem, 2) solving a problem and explaining to a teacher how you did it, or 3) completing a Scratch Card set. The partnering school obtained individual robotic kits at the end of the study for youth who participated in the course and earned a pre-determined amount of computer chip points.

Some degree of participant mortality and attrition was expected due to the fundamental unpredictability of research with human subjects, combined with the highly variable nature of families' schedules during the summer months. In the experimental group, the average attendance was only 65% of instructional days, resulting in an average of 15.75 hours of experience with Scratch and the CC curriculum activities. Only two children attended 100% of instructional days for the experimental group. In the control group, participants on average attended 72.5% of instructional days, resulting in an average of 21.75 hours of experience with Scratch and the CC curriculum activities. Six children in the control group attended 100% of instructional days. On every assessment included in the study, there were fewer participants in the control group that completed a

given measure. While seven participants in the experimental group completed 100% of assessments, only four did so in the control group. The main reason for differences in experimental and control group attendance was that four youth in the control group were present only for the initial portion of the camp and, therefore, only participated in assessments at time point one. Consequently, these children did not receive any instruction or have any experience in the CC curriculum, and could neither complete the PCK<sub>2</sub>, nor were they present to complete remaining assessments.

### Data Collection

For the WJ-IV CF<sub>1</sub> & <sub>2</sub> assessments, the obtained split-test raw scores were converted to total raw scores by calculating the expected total raw score for the full, standard version of the WJ-IV CF using the ratio of split-test raw scores to the maximum amount possible for each time point. The total raw score for each time point was then converted to W scores using standard scoring protocol and software. Performance was reported as W scores, which are a special transformation of the Rasch ability scale (Rasch, 1960; Wright & Stone, 1979), because of the short test-retest time interval of the study, and the mathematical properties of W scores that make them more sensitive to change than the traditionally reported standard scores. The W scale for each subtest of the WJ-IV is centered on a value of 500, which is set to approximate the average performance of 10-year-old individuals (Mather & Wendling, 2014). Youth ranged from 468 to 546 in their W scores across WJ-IV CF measures at both time points, and there was evidence for good reliability between the pre and post-tests, as performance on the WJ-IV CF<sub>1</sub> was highly correlated with performance on the WJ-IV CF<sub>2</sub> ( $r_s = .751, p = .001$ ). Across both experimental and control groups, there were five participants who



took the pre-test, but did not take the post-test, and there were two participants who only took the post-test. Three participants completed the pre-test at time point two.

The KTEA-3 MCA Forms A and B were administered and scored according to standardized procedures, and reported as standard scores rather than W scores, as these were not able to be generated for this assessment. Youth ranged from 63 to 123 in their standard scores across both forms A and B of the KTEA-3 MCA, and as with the WJ-IV CF pre and post-test, there was good evidence for reliability on both forms A and B of the KTEA-3 MCA, as performance on form A was highly correlated with performance on form B ( $r_s = .928, p = .000$ ). Across experimental and control groups, there were five participants who completed form A but not form B; three participants who completed form B but not form A; and one participant who did not complete either forms A or B.

Ratings for participant responses on each item of both the CPS<sub>1</sub> and CPS<sub>2</sub> were provided by three research assistants on a 1-5 Likert type scale (1 = Very Low, 2 = Low, 3 = Average, 4 = High, 5 = Very High) across two dimensions of creativity, i.e., originality and usefulness. Ratings were averaged across raters for each dimension, and then dimensional scores were averaged to create a total score. When one or more raters indicated that the youth may have misunderstood or misinterpreted items, those items were not included in either the overall score, or their overall dimensional scores. Across both experimental and control groups, there were seven participants who completed the CPS<sub>1</sub> and not the CPS<sub>2</sub>; two participants who completed the CPS<sub>2</sub> and not the CPS<sub>1</sub>; and two participants who completed neither the CPS<sub>1</sub> nor the CPS<sub>2</sub>. Dimensional and total scores on the CPS<sub>1</sub> ranged from 2.00 to 4.08, and from 1.33 to 3.83 on the CPS<sub>2</sub>. There

was evidence for good reliability for ratings of responses on the CPS<sub>1</sub> and CPS<sub>2</sub>, as the total scores for each assessment were highly correlated ( $r_s = .712$ ,  $p = .009$ ).

To examine whether raters' interpretation of participant responses tended to agree with one another, interrater reliability was calculated across items and dimensions for both the CPS<sub>1</sub> and CPS<sub>2</sub> using a two-way mixed effects model to test absolute agreement among raters ( $k = 3$ ). Intraclass correlation coefficients (ICC), their 95% confidence intervals, and significance levels are presented in Tables 3.4 and 3.5 below. Overall, there was little evidence to support agreement among raters, as most ICCs were below the 0.5 level, and were non-significant. On seven out of twenty four reported dimensional and total scores across items and assessments, ICCs were negative in value, suggesting that raters actually disagreed in their ratings of creativity across dimensions and items. There were only four significant ICCs across dimensional and total scores for each item on both the CPS<sub>1</sub> and CPS<sub>2</sub>. ICCs that were significant at the  $p < .05$  level ranged from .465 to .659 in value and included the originality dimensional score from item one from the CPS<sub>1</sub> and item two from the CPS<sub>2</sub>, as well as the usefulness dimensional and total score for item four from the CPS<sub>2</sub>. The low number of significant ICCs indicates that raters did not view responses to CPS item prompts as they relate to the constructs of originality and usefulness in a similar way, which may have implications for future studies using the CAT method to assess creativity.

Table 3.4

*Creative Problem Solving Pre-test (CPS<sub>1</sub>) Interrater Reliability Statistics*

		<b>Intraclass Correlation</b>	<b>95% Confidence Interval</b>		<b>Sig.</b>
			<i>Lower Bound</i>	<i>Upper Bound</i>	
<b>Item 1</b>	Originality	.659	-.002	.915	.029*
	Usefulness	-.907	-2.290	.404	.946
	<i>Total</i>	-.376	-2.665	.638	.686
<b>Item 2</b>	Originality	-.441	-1.774	.383	.818
	Usefulness	-.752	-3.161	.330	.877
	<i>Total</i>	-.445	-1.956	.404	.803
<b>Item 3</b>	Originality	.476	-.234	.809	.071
	Usefulness	.322	-.357	.730	.146
	<i>Total</i>	.478	-.242	.810	.072
<b>Item 4</b>	Originality	.027	-1.026	.589	.455
	Usefulness	.351	-.221	.709	.095
	<i>Total</i>	.258	-.440	.674	.194

Table 3.5

*Creative Problem Solving Post-test (CPS<sub>2</sub>) Interrater Reliability Statistics*

		<b>Intraclass Correlation</b>	<b>95% Confidence Interval</b>		<b>Sig.</b>
			<i>Lower Bound</i>	<i>Upper Bound</i>	
<b>Item 1</b>	Originality	-.058	-1.052	.767	.509
	Usefulness	.046	-3.015	.889	.439
	<i>Total</i>	.173	-1.717	.865	.363
<b>Item 2</b>	Originality	.578	.019	.860	.022*
	Usefulness	-2.143	-10.973	.124	.963
	<i>Total</i>	.026	-1.197	.672	.454
<b>Item 3</b>	Originality	-.109	-1.361	.700	.551
	Usefulness	.494	-.196	.871	.064
	<i>Total</i>	.457	-.278	.862	.093
<b>Item 4</b>	Originality	.300	-.246	.736	.125
	Usefulness	.538	-.035	.849	.007*
	<i>Total</i>	.495	-.068	.829	.008*

The variability in ICCs was in part due to the variable number of cases included in the analysis across items, as there were a number of participants who were thought to have misinterpreted or misunderstood items across items on each assessment; subsequently, ratings for these items were not reported or included in calculating interrater reliability. Table 3.6 displays the number of raters who indicated misinterpretation or misunderstanding across items of the CPS<sub>1</sub> and CPS<sub>2</sub>. In general, there was little consensus among raters as it pertained to perceived misinterpretation of item prompts, as evidenced by the trend on both the CPS<sub>1</sub> and CPS<sub>2</sub> for the majority of misinterpretation indications coming from only one rater.

Table 3.6

*Perceived Misinterpretations Across Items on CPS<sub>1</sub> and CPS<sub>2</sub>*

		Number of raters indicating misinterpretation			
		One	Two	Three	Total
<b>CPS<sub>1</sub></b> <b>(n = 20)</b>	Item 1	4	5	1	10
	Item 2	2	0	0	2
	Item 3	11	0	0	11
	Item 4	0	0	0	0
	<i>Subtotal</i>	17	5	1	23
<b>CPS<sub>2</sub></b> <b>(n = 15)</b>	Item 1	6	3	0	9
	Item 2	3	0	0	3
	Item 3	5	2	0	7
	Item 4	2	1	0	3
	<i>Subtotal</i>	16	6	0	22
<b>Grand Total</b>		<b>33</b>	<b>11</b>	<b>1</b>	<b>45</b>

On the PCK<sub>1</sub>, performance was reported according to the scoring procedures of the 2016 USA Bebras Challenge. Each item's level of difficulty was weighted differently, such that correct answers on items categorized as level A difficulty were awarded six

points; nine points for level B difficulty; and 12 points for level C difficulty. Incorrect answers resulted in a two, three, or four-point deduction for difficulty levels A, B, and C respectively, and all additions or deductions started from a base value of 45 points. The minimum score possible was, therefore, zero points as the assessment included five items across each level of difficulty. Total scores for the PCK<sub>1</sub> ranged from 0 to 108, and on average, participants answered 3.3 items correctly, scoring an average of 38.2 points.

For the PCK<sub>2</sub> the total number of items answered correctly was reported, and there was a maximum of seven points possible. Scores ranged from 0 to 4 points, with an average of 1.8 points across both experimental and control groups. Across both experimental and control groups, there were six participants who completed the PCK<sub>1</sub> and not the PCK<sub>2</sub>; one participant who completed the PCK<sub>2</sub> and not the PCK<sub>1</sub>; and four participants who completed neither the PCK<sub>1</sub> nor the PCK<sub>2</sub>.

Participant responses on the Type T questionnaire were assigned a numeric value and summed to create a total score. When participants circled, “Never,” this was assigned a value of zero; when they circled “A Little Bit,” this was assigned a value of one; and when they circled, “A Lot,” this was assigned a value of two. The maximum possible score was thus 16, and scores ranged from 7 to 16 with an average of 10.7. Across both experimental and control groups, there were three participants who did not complete the Type T questionnaire, and they were all in the control group.

Responses on the Computer Usage questionnaire (CUQ) were quantified according to item content, while written responses were qualitatively analyzed. Twenty-one participants took the survey, and all three children who did not complete it were in the control group. Sixty-two percent of youth (n =13) said they did not have a desktop at

home, but had at least one laptop at home (six children with one, and seven children with two). Over half (56%) of participants in the control group reported having a desktop computer at home, while only one quarter of participants in the experimental group reported having a desktop computer at home. Eighty-six percent of youth had at least one tablet at home, with an average of two tablets, and ranging up to five tablets. Ninety percent (n = 19) of youth reported having internet access at home, with twelve usually using a phone to access internet; four usually using another device (console or TV); three usually using laptop; and one usually using tablet. Tables 3.7 and 3.8 display the rates of computer activities in the home and school environments for youth in both experimental and control groups. Participants reporting “other” activities wrote in responses that could generally be categorized into one of the response choices.

Table 3.7

*Youth Computer Activities at Home*

	<b>Experimental (n = 12)</b>	<b>Control (n = 9)</b>
Playing games	100%	58%
Doing homework	67%	42%
Writing	8%	25%
Watching Videos	83%	50%
Reading news stories or articles	25%	17%
Going on social media	58%	42%
Other	42%	8%

Table 3.8

*Youth Computer Activities at School*

	<b>Experimental (n = 12)</b>	<b>Control (n = 9)</b>
Playing games	83%	58%
Doing work	92%	58%
Learning programs or software	58%	58%
Making presentations	33%	25%
Making animations	17%	17%
Making games	8%	17%
Writing computer code	67%	17%
Other	33%	0%

Youth reported on average that they used a smart phone two to three times per day; a tablet one to two times per week; a laptop computer one to two times per week; a desktop computer one to two times per month; “other” computing devices two to three times per day (most commonly a console or TV). Participants in the experimental group reported using their smart phones more often than participants in the control group, but there were no differences in frequency of use of other computing devices between groups. Thirty-eight percent (n = 8) of youth said they had used computer programming or coding to solve a problem or create something in the past. Twenty-four percent (n = 5) of youth said that learning about how computers help to solve problems was not important, or were unsure if it was important. Written responses for interest in using computers to help solve problems focused around using computers to do work, or a vague sense of helping in some way. Forty-three percent (n = 9) of youth said they had participated in a coding camp or club before, with no differences across experimental and control groups, and forty-seven percent (n = 10) said they had done programming or coding on computers before, with several youth reporting that they had used Code.org or

Scratch in the past. Participants in the experimental group reported a higher rate (63%) of previous coding or programming experience than the control group (44%).

There were a number of correlations among responses on items of the Computer Usage Questionnaire (CUQ) and assessment variables. First, youth who reported having access to the internet tended to do better on the PCK<sub>1</sub> ( $r_s = .518$ ,  $p = .023$ ). Second, having a desktop computer in the home was correlated with performance on the WJ-IV CF<sub>1</sub> ( $r_s = .462$ ,  $p = .040$ ), and frequency of desktop use was also positively correlated with performance on the WJ-IV CF<sub>1</sub> ( $r_s = .453$ ,  $p = .045$ ). Third, frequency of laptop use was positively correlated with performance on the KTEA-3 MCA<sub>B</sub> ( $r_s = .544$ ,  $p = .036$ ). Lastly, reported tablet usage was positively correlated with several assessment variables, indicating that youth who reported using tablets more often tended to do better on various assessments included in the study. Tablet use was correlated with performance on the WJ-IV CF<sub>1</sub> ( $r_s = .657$ ,  $p = .002$ ), the KTEA-3 MCA<sub>A</sub> ( $r_s = .671$ ,  $p = .002$ ) and KTEA-3 MCA<sub>B</sub> ( $r_s = .552$ ,  $p = .033$ ), as well as the CPS<sub>1</sub> ( $r_s = .483$ ,  $p = .042$ ) and CPS<sub>2</sub> ( $r_s = .574$ ,  $p = .032$ ). Participants who indicated that they had participated in a coding camp or club, or had previously engaged in computer programming or coding activities were not associated with significantly better scores across assessments, and in fact, the only significant correlation with any assessment variable (PCK<sub>2</sub>) and prior experience with computer programming or coding was negative in value ( $r_s = -.658$ ,  $p = .028$ ).

The number of participants in the experimental and control groups who completed assessments, their mean scores, and standard deviations are displayed in Table 3.9.



Table 3.9

*Descriptive Statistics Across Assessments by Group*

	Experimental (n = 12)			Control (n = 12)			Total (n = 24)		
	<i>N</i>	<i>Mean</i>	<i>SD</i>	<i>N</i>	<i>Mean</i>	<i>SD</i>	<i>N</i>	<i>Mean</i>	<i>SD</i>
<b>WJ-IV CF<sub>1</sub></b>	12	490	13.9	10	497	22.9	22	493	18.3
<b>WJ-IV CF<sub>2</sub></b>	10	490	11.5	8	490	10.9	18	490	10.9
<b>KTEA-3 MCA<sub>A</sub></b>	12	87	17.0	7	92	10.6	19	89	14.8
<b>KTEA-3 MCA<sub>B</sub></b>	9	88	15.6	8	90	18.0	17	89	16.3
<b>CPS<sub>1</sub></b>	11	3.03	0.44	9	2.98	0.45	20	3.00	0.43
<b>CPS<sub>2</sub></b>	9	2.66	0.55	6	2.59	0.35	15	2.63	0.47
<b>PCK<sub>1</sub></b>	11	36.7	28.7	8	40.3	19.0	19	38.2	24.5
<b>PCK<sub>2</sub></b>	7	1.71	1.1	6	1.83	1.3	13	1.77	1.2
<b>Type T</b>	12	9.9	2.2	9	11.8	2.2	21	10.7	2.3

## Relationship Among Variables

There were a number of variables that showed significant correlations with one another among participants in both the experimental and control groups, and as a whole. These relationships were produced using Spearman's *rho* ( $r_s$ ) with cases excluded pairwise at both the group and overall sample levels, as the low overall N in the study necessitated the use of nonparametric statistics. Demographic variables (age, sex, PSSA scores, special education status, Type-T personality characteristic), classroom variables (attendance, instructional hours, total chip count), and assessment variables (WJ-IV CF<sub>1</sub> & 2, KTEA-3 MCA<sub>A</sub> & B, CPS<sub>1</sub> & 2, and PCK<sub>1</sub> & 2) were entered into the analyses. Correlation statistics are primarily reported in terms of the overall sample, as not only were there

fewer participants in the control group on average who completed various assessments, and thus, included in their respective group analyses across those variables ( $n = 10$  for the experimental group and  $n = 7.67$  for the control group), but also because the group correlations generally followed the same trends as the overall sample correlations. There were, however, several interesting differences in group correlations, and they are discussed first, followed by presentation of the relationship among variables in the overall sample. Full correlation matrices for all variables entered into the analyses by overall sample can be found in Appendix F, and by groups in Appendices G and H.

The first difference between correlations in the experimental and control groups was that there were overall less significant correlations in total for the control group, with only nine significant correlations emerging in the analysis of 16 variables, while there were 27 significant correlations in the experimental group. Second, special education status was negatively correlated with the Type T personality characteristic in the experimental group ( $r_s = -.596$ ,  $p = .041$ ), but positively correlated with the Type T personality characteristic in the control group ( $r_s = .681$ ,  $p = .043$ ). While these correlations are notable, they require some qualification, as there were only four participants in total who were classified as receiving special education across both groups. Therefore, upon closer inspection, these correlations suggest that the two children in the experimental group who received special education in school tended to report higher levels of thrill seeking behavior, while the two children receiving special education in the control group tended to report lower levels of thrill seeking behavior. Lastly, while participants in the control group who obtained higher scores on a pre-test of CT skills ( $PCK_1$ ) were associated with earning more computer chips during the CC

course ( $r_s = .800$ ,  $p = .031$ ), participants in the experimental group whose academic achievement, as measured by PSSA ELA and PSSA Math scores, were associated with earning more computer chips. These correlations may partly be explained by the additional effort made by instructors to reinforce youth in the experimental group who had difficulty with programming activities, perhaps due to a relationship between academic achievement and success with CC activities; whereas youth in the control group who possessed higher CT skills prior to beginning CC lessons and activities were able to experience more success, and subsequently earned more computer chips.

Analyzed as a whole, and independent of the CC course instruction, being a male was moderately correlated to higher performance on the PCK<sub>1</sub> ( $r_s = .485$ ,  $p = .022$ ). Girls tended to perform worse than boys on PSSA ELA ( $r_s = -.405$ ,  $p = .050$ ). Youth in higher grade levels tended to perform better on the CPS<sub>2</sub> assessment ( $r_s = .680$ ,  $p = .005$ ), but there was no relation to grade level and the CPS<sub>1</sub> assessment ( $r_s = .252$ ,  $p = .298$ ).

There were significant positive correlations among scores on both the Math and ELA PSSAs, the KTEA MCA Forms A and B, and CPS assessments, and they are displayed in Table 3.10. The correlations between the PSSA ELA and KTEA-3 MCA<sub>A</sub> and KTEA-3 MCA<sub>B</sub> were lower in value than the correlations between the PSSA Math and KTEA-3 MCA<sub>A</sub> and KTEA-3 MCA<sub>B</sub>, and this suggests good concurrent validity for the measure of mathematic ability on both the Math PSSA and KTEA-3 MCA assessments. Participant PSSA ELA and Math scores were also positively and moderately correlated to CPS<sub>1</sub> and CPS<sub>2</sub> scores, with correlation coefficients ranging from .488 to .544.

Table 3.10

*Correlations among PSSA, KTEA-3 MCA, and CPS scores*

	1	2	3	4	5	6
<b>1. PSSA ELA</b>	—					
<b>2. PSSA Math</b>	.533**	—				
<b>3. KTEA-3 MCA<sub>A</sub></b>	.462*	.786**	—			
<b>4. KTEA-3 MCA<sub>B</sub></b>	.607**	.831**	.928**	—		
<b>5. CPS<sub>1</sub></b>	.488*	.526*	.799**	.788**	—	
<b>6. CPS<sub>2</sub></b>	.544*	.518*	.655*	.681**	.712**	—

Note: \*  $p < .05$ ; \*\*  $p < .01$

While the correlations among standardized measures of academic achievement were expected given their ubiquitous usage in the field of education, and known strong psychometric properties, the strong correlation between the CPS<sub>1</sub> and CPS<sub>2</sub> ( $r_2 = .712$ ,  $p = .009$ ) was particularly notable, as these instruments were developed specifically for this study. This finding suggests good reliability of the CPS assessment in the context of the current study.

The total number of token computer chips earned by each participant across the duration of the study was summed and used for all analyses. Overall, the experimental group earned more chips on average than control group (100 vs. 24), but there were fewer participants in the control group who attended instructional days regularly. Taking this into account by multiplying the percentage of days attended by each participant with their total chip counts, the trend remained the same. Participants in the experimental group earned an average of 7.4 chips by attendance, while those in the control group earned an average of 2.7 chips by attendance. Unsurprisingly, youth who attended more frequently earned more chips, and there was a strong correlation between the attendance and the total number of chips earned throughout the course of the study ( $r_s = .745$ ,  $p = .000$ ).

Cumulative chip count was also negatively correlated with scores on the Type T personality questionnaire ( $r_s = -.513, p = .035$ ), suggesting that youth who reported more risk-taking behavior tended to earn less chips. This was the only significant correlation with any demographic or assessment variable and the Type T personality questionnaire.

One interpretation of this correlation may be that the Type T personality characteristic could be associated with more emphasis on nonconformity, suggesting that youth with a higher risk-taking or thrill-seeking personality profile would be less likely to subscribe to a positive behavior reinforcement system. There were, however, some problems with the chip system itself, as earning a computer chip was not entirely dependent on the group rules laid out in the beginning of the course. On the one hand, while there was a group of youth who genuinely earned chips for completing Scratch cards, helping others to resolve an issue in Scratch, or showing an instructor their completed work; there was, on the other hand, another group of youth who seemed to become disinterested with lessons and the Scratch programming environment as they found the activities somewhat frustrating, and subsequently did not earn token computer chips. These participants who expressed more frustration and reservation in completing lessons and projects ended up earning chips more easily as instructors felt the need to increase the frequency of reinforcement by reducing the criteria needed to earn chips on an individual basis. Viewing the total chip count with this observation in mind, another interpretation of the negative correlation with Type T questionnaire scores may be that children who obtained higher scores on the Type T personality questionnaire earned less computer chips because there was not enough thrill or excitement in the computer programming activities comprising the CC curriculum.

There were several significant correlations between the PCK<sub>1</sub> and PCK<sub>2</sub>, and other assessments. First, performance on the PCK<sub>1</sub> assessment was positively correlated with their performance on the both the WJ-IV CF<sub>1</sub> ( $r_s = .534, p = .022$ ), and WJ-IV CF<sub>2</sub> ( $r_s = .559, p = .024$ ). Similarly, performance on the KTEA-3 MCA<sub>A</sub> ( $r_s = .578, p = .015$ ) and KTEA-3 MCA<sub>B</sub> ( $r_s = .812, p = .000$ ) was also correlated with their performance on the PCK<sub>1</sub> assessment. These correlations suggest that inductive reasoning, and knowledge of mathematical concepts is associated with successfully solving tasks from the Bebras Challenge CT assessment. This makes intuitive sense, as the tasks on the PCK<sub>1</sub>, WJ-IV CF, and KTEA-3 MCA assessments involved recognizing an underlying rule or pattern, and applying mathematical knowledge.

On standardized norm-referenced assessments included in the study, there were also several correlations that arose. First, performance on the WJ-IV CF<sub>1</sub> was positively correlated with ratings on the CPS<sub>1</sub> ( $r_s = .510, p = .026$ ), but not significantly with the CPS<sub>2</sub>. The WJ-IV CF<sub>2</sub> was not correlated with either the CPS<sub>1</sub> ( $r_s = .119, p = .672$ ) or the CPS<sub>2</sub> ( $r_s = .195, p = .504$ ). There appears to mixed evidence for a consistent link between problem solving as measured by the WJ-IV Concept Formation subtest, and ratings of creativity as measured by the CPS pre and post-tests. Finally, participants who obtained higher scores on the KTEA-3 MCA Forms A and B tended to obtain higher ratings on both the CPS<sub>1</sub> and CPS<sub>2</sub> assessments, with correlation coefficients ranging from .681 to .799. These results aligned with the aforementioned correlations with participant PSSA performance and the CPS ratings, suggesting that participants who had greater mathematical conceptual knowledge generally were thought to respond in a more original and useful way to CPS item prompts.

## CHAPTER 4

### RESULTS

The current study aimed to identify the degree to which youth who participated in computer programming activities in the Scratch environment in a summer camp program 1) demonstrated measurable changes in problem solving ability and creative thinking, and 2) learned and applied computational thinking skills. The overall low N in each group required the use of nonparametric statistics for analyses; specifically, the Mann-Whitney *U* Test to explore group differences across the WJ-IV CF, KTEA-3 MCA, and CPS pre and post-tests; and Spearman's *rho* ( $r_s$ ) to examine the relationship between demographic and assessment variables.

#### RQ1: Problem Solving and Creative Thinking

Results from independent samples Mann-Whitney *U* tests showed that the mean ranks of all assessment variables included in the analysis (WJ-IV CF<sub>1 & 2</sub>, KTEA MCA<sub>A & B</sub>, and CPS<sub>1 & 2</sub>) did not differ significantly across experimental and control groups, supporting retention of the null hypothesis. Group differences across assessments were analyzed in addition to change scores across each assessment. Results are displayed along with the N for each analysis, the mean rank, Mann-Whitney *U* value, p-value, and effect size when appropriate in Tables 4.1 and 4.2 below.

Table 4.1

*Mann-Whitney U Test Statistics Across Assessments and Group*

	Experimental		Control		Mann-Whitney <i>U</i>	Sig. (1-tailed)
	<i>N</i>	Mean Rank	<i>N</i>	Mean Rank		
WJ-IV CF <sub>1</sub>	12	10.88	10	12.25	52.5	.318
WJ-IV CF <sub>2</sub>	10	8.85	8	10.31	33.5	.292
KTEA MCA <sub>A</sub>	12	9.17	7	11.43	32.0	.210
KTEA MCA <sub>B</sub>	9	9.00	8	9.00	36.0	.509
CPS <sub>1</sub>	11	9.82	8	10.25	42.0	.444
CPS <sub>2</sub>	9	8.22	6	7.67	25.0	.420

Table 4.2

*Mann-Whitney U Test Statistics for Change in Assessment Scores*

	<i>N</i>	Mann-Whitney <i>U</i>	Sig. (1-tailed)	$\eta^2$
WJ-IV CF $\Delta$	16	27.0	.792	.007
KTEA MCA $\Delta$	14	22.0	1.00	.000
CPS $\Delta$	12	7.50	.154	.191

To further illustrate the lack of difference between groups across CPS assessments, Figure 4.1 displays the mean ratings of participants in both the experimental and control groups across dimensional and total scores for both the CPS<sub>1</sub> and CPS<sub>2</sub>. Participants tended to provide responses to the CPS<sub>2</sub> that were rated as less creative in both the dimensions of usefulness and originality.



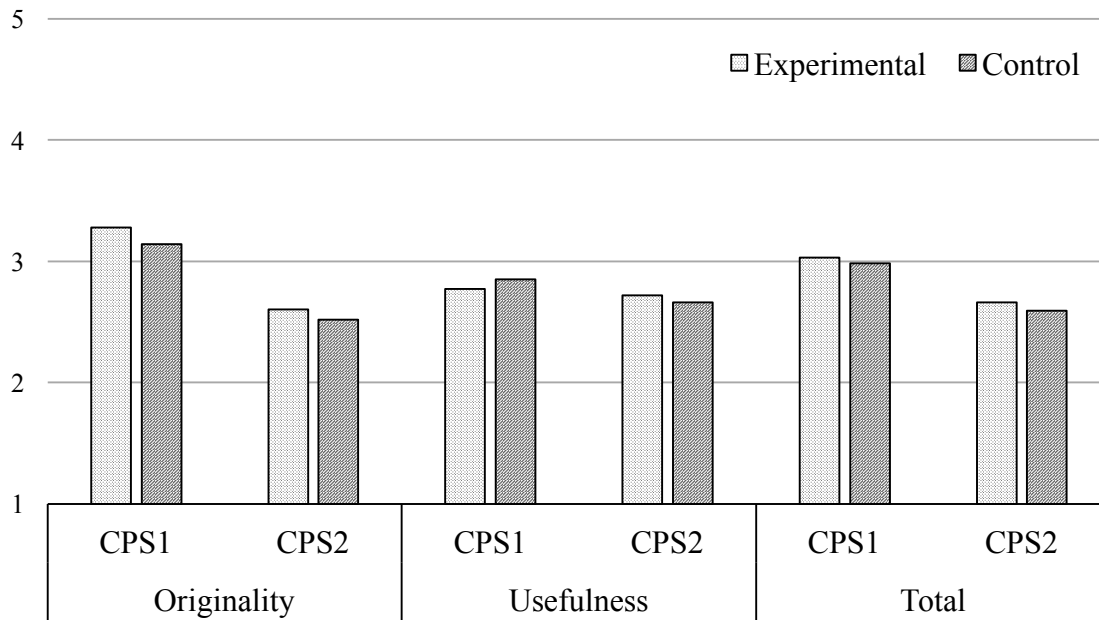


Figure 4.1. Mean ratings on the CPS assessments

#### RQ2: Computational Thinking Skills

The conceptual framework underlying computational thinking skills assessed on both the PCK<sub>1</sub> and PCK<sub>2</sub> differed, so there was no direct method to test whether participants learned specific CT skills measured on the PCK<sub>1</sub> from results on the PCK<sub>2</sub>; rather, to investigate whether participants who took both the PCK<sub>1</sub> and PCK<sub>2</sub> (n = 12) made any meaningful gains in CT skills after participating in the computer course, PCK<sub>1</sub> scores were assigned a rank value, and their ranked performance was then compared to their total scores on the PCK<sub>2</sub>. Figure 4.2 displays the ranked order of participant's performance on the PCK<sub>1</sub> and their total PCK<sub>2</sub> scores to graphically illustrate that there was no clear trend in whether participants who did or did not perform relatively well on the PCK<sub>1</sub> made gains or losses on the PCK<sub>2</sub>. A higher ranking denotes a lower score such that the "1" on the x-axis was the highest score, and the "12" was the lowest score.

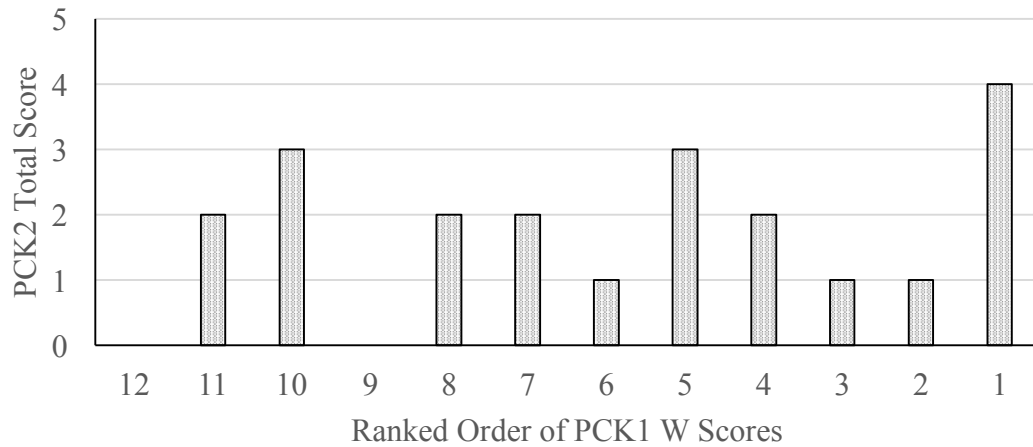


Figure 4.2. Ranked Order of PCK<sub>1</sub> Scores by PCK<sub>2</sub> Scores Across Participants

Total W scores on the PCK<sub>1</sub> were not correlated with total scores obtained on PCK<sub>2</sub> ( $r_s = .289$ ,  $p = .455$ ), but there was some support for the difficulty level categorization of the PCK<sub>1</sub> assessment, as the total number of correct answers across participants who completed the PCK<sub>1</sub> assessment ( $n = 19$ ) on items in difficulty level A was 24/95, 27/95 for level B; and only 12/95 for level C. In other words, participants tended to answer fewer of the most difficult items than the least difficult items on the PCK<sub>1</sub>. Figures 4.3 and 4.4 display the number of participants across both experimental and control groups that answered items of the PCK<sub>1</sub> and PCK<sub>2</sub> correctly.

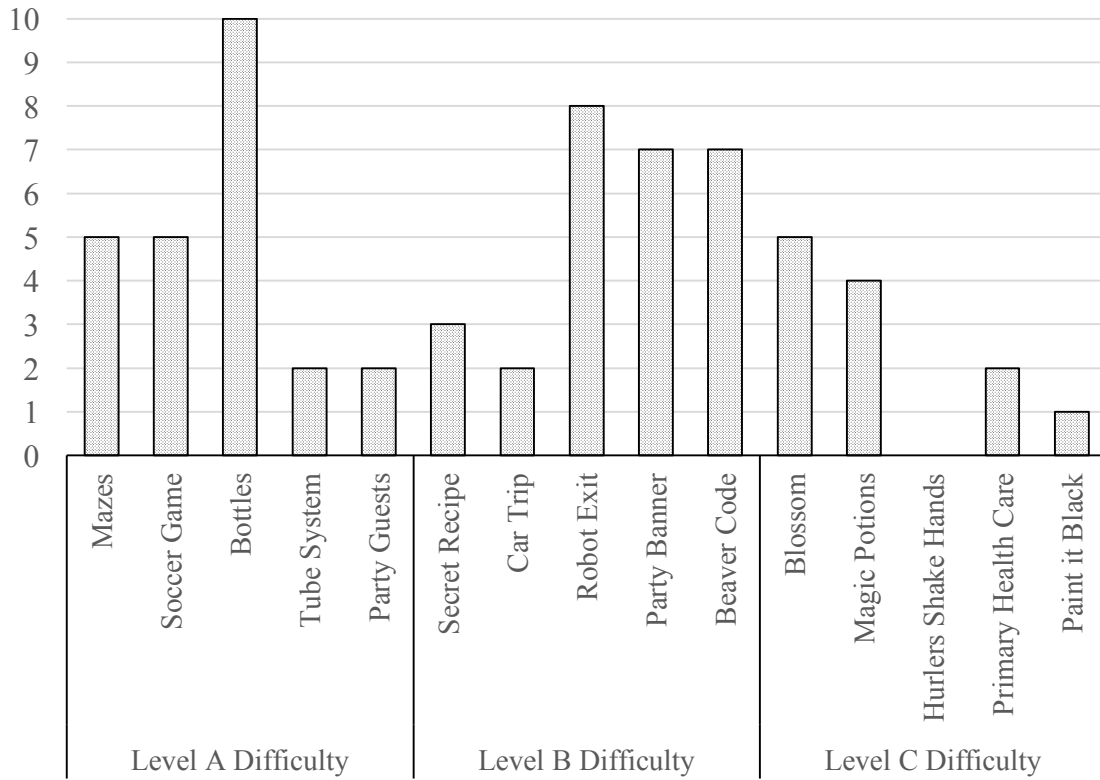


Figure 4.3. Number of Correct Responses Across PCK<sub>1</sub> Items

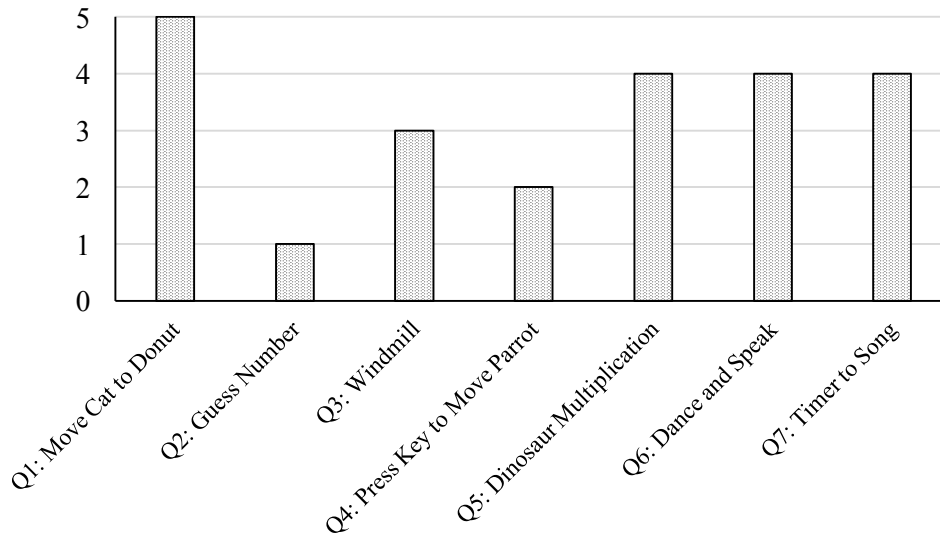


Figure 4.4. Number of Correct Responses Across PCK<sub>2</sub> Items

To understand whether participants who answered certain questions correctly on the PCK<sub>1</sub> assessment tended to obtain correct responses on specific items from the PCK<sub>2</sub> assessment, items on each PCK assessment were correlated with one another. Using this method, correlations could be indicative of whether participants retained any prior-existing CT skills required to correctly answer items on the PCK<sub>1</sub>, or whether items on the PCK<sub>1</sub> assessed similar CT skills on PCK<sub>2</sub>. Items that showed a significant correlation using Spearman's *rho* ( $r_s$ ) were then compared to one another across the CT skills and key programming concepts, as well as specific curricular content embedded within each item. There were four significant correlations found among items on the PCK<sub>1</sub> and PCK<sub>2</sub> for which the details are presented below.

There were two identical, significant correlations with the PCK<sub>1</sub> Soccer Game item, and questions one (Move Cat to Donut) and five (Dinosaur Multiplication) from the PCK<sub>2</sub> ( $r_s = .625$ ,  $p = .03$ ). The CT skills and key programming concepts assessed in the Soccer Game item from the PCK<sub>1</sub> were algorithmic thinking, evaluation, and "IF" condition, while the CT skills and curricular content assessed in question one from the PCK<sub>2</sub> were sequences, and understanding of rotation, direction, and spatial orientation. The CT areas and curricular content assessed in question five of the PCK<sub>2</sub> were sequences, loops, operators, data, and understanding of the ">" and "\*" symbols, as well as understanding the concept of a variable.

There was a perfect correlation between the Party Guests item of the PCK<sub>1</sub> and question two (Guess Number) on the PCK<sub>2</sub> ( $r_s = 1.00$ ). The reason for this perfect correlation was that there was only one participant who responded correctly on question two of the PCK<sub>2</sub>, and this participant also responded correctly to the Party Guests item

on the PCK<sub>1</sub>. There was also one participant who answered the Party Guests item of the PCK<sub>1</sub> correctly, but did not complete the PCK<sub>2</sub> assessment. The CT skills and key programming concepts assessed in the Party Guests item of the PCK<sub>1</sub> were algorithmic thinking, decomposition, dependency, and graphs. The CT skills and curricular content assessed in question two of the PCK<sub>2</sub> were conditionals, operators, data, and understanding of the “>” symbol, as well as understanding of the “ask” and “say” Scratch blocks.

Finally, there was a significant correlation between the Blossom item of the PCK<sub>1</sub> and question four (Press Key to Move Parrot) of the PCK<sub>2</sub> ( $r_s = .632$ ,  $p = .027$ ). The CT skills assessed in the Blossom item of the PCK<sub>1</sub> were evaluation and generalization, while the CT skills assessed in question four of the PCK<sub>2</sub> were events, and understanding of the  $x$  and  $y$  axes.

## CHAPTER 5

### DISCUSSION

In lieu of the number of factors that limit the internal and external validity of the results, there were several notable findings that emerged, and may be important for future researchers and educators interested in the study of how and what ten to fourteen-year-old children learn through computer programming activities, or for those interested in implementing the Creative Computing Curriculum. A discussion of important findings and their relation to future research or instructional endeavors is presented first, followed by a discussion of the limitations of the study.

#### Notable Findings

##### *No group differences across assessment variables*

The results of Mann-Whitney  $U$  tests to answer the primary research question showed that there were no statistically significant differences between experimental and control groups in measures of problem solving (the WJ-IV CF and KTEA-3 MCA assessments) and creativity after participating in computer programming activities. In other words, ten to fourteen-year-old children who were provided with direct instruction, and who explored the Scratch programming environment through the Creative Computing (CC) Curriculum in a roughly two-week summer computer course, did not perform any better or worse on measures of problem solving skills and creative thinking ability than children who did not participate in similar activities during the same time period. The lack of any significant changes in problem solving or creative thinking skills between groups may be due in part to the short timespan over which youth engaged in computer programming activities, and also because youth explored and created with

Scratch at an introductory, or surface level, as opposed to a more in-depth and complex level. Previous studies showing gains in problem solving or creative thinking skills often involved larger treatment dosages over longer timespans, allowing youth to spend more time planning, developing, testing, and personalizing their projects. In the current study, children largely followed steps laid out in the CC Curriculum, and rarely reached a level of competency to branch off into their own original ideas. Furthermore, when they encountered malfunctions in their projects, they tended to quickly ask for help, or resorted to engaging in other, non-Scratch activities, rather than effectively engaging in a problem solving process.

Another factor influencing the lack of any detectable learning gains on selected measures relates to the decision to exclude design journals recommended in the CC curriculum for the current study. As a result, many children either relied heavily on following the step-by-step procedures to complete CC lessons and activities, or strayed entirely away from the designated lessons, choosing to explore the Scratch environment in a somewhat indiscriminate fashion. Had there been more time devoted to emphasizing the importance of the planning and design phase in for individual projects, participants may have more effectively been able to conceptualize the steps and associated computational processes involved in realizing their desired outcomes. A theme discovered early on in the experimental group was that while participants expressed a desired goal for their Scratch project, they had trouble delineating the steps necessary in order to put together functional Scratch programs to reach their goals. One-on-one assistance provided by instructors often included a breakdown of the larger goal into smaller steps, and then translating those steps into Scratch code. Sometimes, this

individual assistance included math calculations, diagrams to explain the Cartesian coordinate system, or explanations of the function of specific code blocks in simplified language. Whether participants would have been able to experience more success in Scratch with greater attention to planning and design, and thus move more quickly through lessons with less frustration is not entirely clear; nonetheless, dedicating more time and energy in the design and planning phase, especially on more open-ended projects, may have allowed participants to better internalize key computational concepts and processes, and should be considered in future research.

Lastly, the quality and pace of instruction between experimental and control groups differed, as feedback from the experimental group was taken into consideration for instruction with the control group, such that unsavory lessons and activities were removed from planned instruction for the control group, and more time was allowed for lessons wherein the experimental group reported high interest and satisfaction. As a result, participants in the control group were able to move more quickly through the CC curriculum, and engage in more complex lessons and activities; but, because the control trial phase of the study was completed after the experimental group had completed the class, results related to the primary research question of whether children who participate in computer programming activities demonstrated changes in problem solving skills and creative thinking were, thus, not affected. Rather, participants in the control group could have been more likely to learn more computational thinking skills through more refined and higher quality instruction and experience with Scratch.



*Difficulty of Teaching the Creative Computing Curriculum in an Informal Learning Environment*

Due to the time of year in which the study took place, i.e., during the school's summer break, many participants seemed to take on a vacation-mindset, such that they became frustrated with having to complete the assessments, and participate in a semi-structured course despite having assented to the study prior to the beginning of summer program. This led to participants hurrying to finish assessments and CC lessons and activities, consequently affecting the validity of the results. The degree to which this summer-break mindset affected the children seemed to be most apparent during the beginning of the study when the daily routine of the summer programming was more novel. The token computer chip, positive reinforcement system was a preemptive attempt to boost motivation, and add extra incentive for the children to participate in computer programming activities. Additionally, the partnering school's purchase of individual robotic kits for which participants earning a pre-determined number of chips could exchange was an attempt to further bolster motivation. These steps, however, did not seem to be entirely effective, particularly in the experimental group, where instructors were required to spend more time and energy responding to challenging attitude and behavior rather than focusing on individual and group instruction. After each instructional day, instructors collaborated to introduce lessons in more relevant ways, incorporating things that the youth expressed liking about the previous day's lesson, and connecting CC lessons to the broader context of computer science and its application to real-world problems. Although participants appeared to take interest in these

supplemental and adjusted activities, they continued to express some frustration and discontent with CC curriculum lessons and activities.

While the CC curriculum was developed to garner high interest and engagement among school age youth, there were a few lessons that did not go over well with participants in the current study, especially in the experimental group. For example, participants in the experimental group reported disliking the “10 Blocks” and “Orange Square, Purple Circle” lessons, and these lessons were subsequently removed or deemphasized from the planned sequence of lessons for the control group. The children seemed to become disinterested in working within the confines of these lessons, appearing to take more interest in creating their own projects with personalized media rather than being limited by rules constricting their use of various types of code or seemingly mundane shapes and objects. Future researchers or educators looking to use the CC Curriculum should keep these observations in mind when planning lessons and activities.

In a further attempt to boost interest and engagement with CC curriculum activities, instructors taught participants how to upload media into Scratch, calculate beats-per-minute for their music video projects, and brainstorm ideas for stories and games. These instructional components were not explicitly part of the CC curriculum, but were well received by the majority of children, and should be considered for educators using the CC curriculum as a framework for teaching CT skills through Scratch in future endeavors. In this way, the CC Curriculum can serve as a launching point, or guide for instructors to use, but in order to effectively engage youth in Scratch programming activities, a degree of improvisation and flexibility is required.

The informal learning environment in which the current study took place, and other similar settings such as after-school enrichment programs, present challenges that should be taken into consideration for engaging youth more deeply with computer programming activities. Youth seem to approach this type of setting differently from the manner in which they might approach similar tasks in a more formal classroom learning environment. The children may have been more likely to engage with curricular material had they attended more regularly, and taken the lessons and tasks more seriously in a more formal, compulsory classroom setting. While some participants seemed to engage more effectively with the Scratch programming environment, and genuinely enjoy aspects of the lessons, others took a more relaxed and nonchalant approach to the material. Additionally, the amount of assessment time seemed to exceed developmental norms for this age group, and participants became frustrated with having to complete the battery of assessments selected for the study. Future studies should keep these observations in mind when selecting assessments, and planning for activities and lessons in this type of learning environment.

The variability of engagement with the CC Curriculum was qualitatively noted in the current study, but not quantitatively analyzed. Future researchers should attempt to understand the process in which children work in the Scratch environment more closely than in the current study. Efforts by some researchers to extract Scratch project data files describing the code content and actions in children's Scratch projects across specified time intervals has been one area in which this process-oriented approach as opposed to a more product-oriented approach is already being implemented (e.g., Fields, Quirke, Amely, & Maughan, 2016; Fields, Quirke, Horton, Maughan, Velasquez, Amely, &

Pantic, 2016; Pantic, Fields, & Quirke, 2016). In the current study, it would have been helpful to more systematically and quantitatively categorize participants into levels of Scratch engagement, and one way this could have been done would have been to introduce a structured observation system that either a researcher or analytic digital tool could use to gather information about what children are actually doing on their computers at specified time intervals by taking “snapshots” of their screens either digitally or in-person.

It may also be helpful for educators and researchers interested in using the CC curriculum or other Scratch-based lessons and activities to get a sense of learners’ knowledge as it relates to general computer literacy prior to beginning instruction. Using a pre-test of computer literacy could be helpful to determine how much time should be spent teaching youth how to download, convert, and save media files from the web, and locate them to upload into the Scratch environment, or understand nuances of both the computer and Scratch interfaces, i.e., keyboard shortcuts, right-clicking, etc. While the Scratch environment does offer a library of pre-loaded media to use, participants in the current study sometimes expressed a desire to incorporate their own media, and when they were taught how to do so (usually with one-on-one instruction), they generally became more interested in the project they were creating. This seemed to result in a more meaningful experience with Scratch, and it is possible that by tapping into the personal appeals of learners, that lessons and activities may become meaningful and engaging for youth. Consequently, youth may be better able to learn CT skills when it is not viewed as just another academic exercise. Of course, understanding what is meaningful to varying populations of children will undoubtedly differ across ages, settings, and regions;

nonetheless, it is an important consideration to make when implementing the CC Curriculum or similar guides.

Similarly, it became evident that many children did not understand some fundamental mathematical concepts required to effectively create projects with their desired functions and outcomes. For example, many participants needed direct instruction on the Cartesian coordinate system, understanding of angles and rotation, recognizing inequalities, and conceptualizing variables. By obtaining a gauge of children's knowledge of these key math concepts necessary for many aspects of Scratch programming prior to instruction, a more targeted approach to teaching necessary mathematics skills could take place, and result in less frustration and more success in Scratch. While the KTEA-3 MCA was used to measure changes in learning in the current study, future studies should first analyze the requisite skills needed for activities, and then pre-test children on these skills in hopes of identifying gaps to be filled before or during instruction.

#### *Difficulty Assessing Computational Thinking*

Results from the PCK assessments to answer the secondary research question were difficult to analyze and interpret given their differing conceptual frameworks, as well as the overlapping of CT skills with specific curricular content within items. Furthermore, the lack of correlation between the PCK pre and post-tests may be indicative of misalignment of skills assessed with each measure, but could also be related to the fundamental conceptual differences of each assessment. It was difficult to know whether participants who answered certain questions on the PCK<sub>1</sub> and PCK<sub>2</sub> assessments correctly were employing the same CT skills, or whether there was an alternative reason that participants tended to answer these questions correctly. For example, the Soccer

Game item from the PCK<sub>1</sub> was correlated with questions one (Move Cat to Donut) and five (Dinosaur Multiplication) on the PCK<sub>2</sub>. Perhaps participants used similar algorithmic thinking and evaluative CT skills in answering the Soccer Game question from the PCK<sub>1</sub> and questions one and five of the PCK<sub>2</sub>, but there was no way of knowing besides analyzing the nature of each task on a granular level post hoc. On the surface, however, there seemed to be few similarities between these questions on both PCK assessments. Without direct observation or interviews with the children, understanding how they managed to think through problems was not possible. A more interactive, computer-based assessment capturing participants' processes-in-action through analysis of their actions as they completed items either through direct observations and interviews, or through JSON file analysis would have been a more valid measure of CT skills, and easier to interpret. Future studies looking to understand how and what children learn as it relates to CT skills and areas should consider this aspect when selecting and designing assessments.

The PCK<sub>1</sub> assessment (the 2016 USA Bebras Challenge) was selected as a pre-test of CT skills and knowledge due to its attempt to assess CT skills without relying on prerequisite syntactical knowledge of specific programming languages, but there have been critiques of its validity in assessing CT skills. For example, Izu et al. (2017) notes that researchers have found a correlation between the length of Bebras Challenge's task and perceived difficulty in elementary students. Similarly, the linguistic demands across items of the Bebras Challenge are thought to be a major barrier to obtaining correct responses (Yagunova, Podznyakov, Ryzhova, Razumovskaia, & Korovkin, 2015). With this notion in mind, one reason why participants may not have correctly answered items

on the PCK<sub>1</sub> could have been that they were not able to understand or comprehend the premise of the question because there were too many factors to retain within their working memory, or the language was too complex. Without a direct measure of working memory or reading comprehension, however, this supposition was unable to be tested in the current study, but may be helpful in future endeavors. Similarly, for the PCK<sub>2</sub>, participants may have been unable to attend to all the details embedded within each item, overlooked key details, or misunderstood mathematical symbols of the item content and response choices. Also, unlike some items of the PCK<sub>1</sub>, the PCK<sub>2</sub> did not allow for immediate feedback, testing, and debugging –a key CT process –due to the fact that it was administered as a paper-pencil test for practical purposes.

The measurement of CT skills on the PCK<sub>2</sub> may also have been confounded by items inclusion of specific curricular content using various Scratch codes and sequences, as opposed to the more theoretical and abstract tasks comprising items of the PCK<sub>1</sub> assessment. Perhaps the PCK<sub>1</sub> would have been a more direct measure of CT skills learned after participating in computer programming activities; however, with this type of design, practice effects would have confounded an identical post-test. Furthermore, within this hypothetical paradigm, learning of specific curricular content could not have been assessed. Relatedly, the items comprising the PCK<sub>2</sub> were either independently developed, or borrowed from the “Coding Quiz” utilized in Straw, Bamford, and Styles (2017), as opposed to validated and normed on a large sample of children. This made generalizing performance on the PCK<sub>2</sub> beyond the context of the current study impossible.

A think-aloud procedure, or post-assessment interview with participants may have been able to shed more light onto the process in which they answered each question, and thus, allowed for comparisons to be made between participants' actual approach to working through the problems, and the authors of the Bebras Challenge suggested approach to working through the problems. This methodology, however, is not without its practical limitations, as transcribing, coding, and qualitatively analyzing participant descriptions of their thought processes is rather labor intensive. Similarly, on the PCK<sub>2</sub>, a think-aloud procedure or post-assessment interview may have resulted in greater insight into the strategies children used to answer each question. This method, however, is limited by the amount of time required to collect and analyze narratives, as well as children's own ability to describe their thought processes given their variable language skills

#### Limitations

The results of the current investigation should be interpreted cautiously as there were a variety of factors that impacted the planned research design, assessment procedures, and classroom instruction. The first and foremost limiting factor was the overall low sample size, as well as the varying number of assessments completed by each participant in both experimental and control groups. Consequently, there were many gaps in pre/post assessments across time-points and participants. Furthermore, there were four participants in the control group who only attended for the initial assessment phase of the study, and two participants in the experimental group who also only attended for the initial assessment phase of the study. Relatedly, attendance was inconsistent, meaning that the amount of instruction and experiential time with the CC curriculum and



associated lessons and activities varied widely across participants within and between groups. The generalization of the findings is, therefore, minimal, and in some ways questionable even within the context of the study.

Another way in which the study may be limited relates to group differences. Although participants demonstrated similar variance in obtained scores across measures, there were two notable differences between participants in the experimental and control groups. First, about twice the rate of participants in the control group reported having a desktop computer at home. Because lessons and activities in the CC curriculum were delivered and explored through desktop computers, youth with desktops at home may have been in a position of more familiarity with the desktop computer interface, resulting in more efficient engagement with the course content than those who did not have desktop computers at home. Second, participants in the experimental group reported a higher rate of previous experience with computer coding or programming than in the control group. Although there were no positive correlations related to previous experience with coding or computer programming and assessment variables either by group or by overall sample, it may have impacted motivation in the experimental group, as the Scratch environment may have been too familiar and subsequently uninteresting to them; thus, lowering their motivation to complete lessons. Alternatively, prior coding or computer programming experience was not quantified or explored beyond a simple “yes” or “no” response on the CUQ questionnaire. Youth may have fundamentally misunderstood what “coding” or “computer programming” activities entailed, or only participated in brief introductory exercises such as Code.org’s popular “Hour of Code” program.

There were also some logistical concerns related to curriculum implementation and assessments selected and developed for the study that also limit the internal validity of the study. First, the amount of instructional time differed for experimental and control groups, as there was a scheduling miscommunication that resulted in the loss of nearly one week of instructional time for the experimental group, and two weather-related school closures that prevented participants in the experimental group from gaining additional experience with CC lessons and activities. Due to this scheduling miscommunication, participants in the experimental group completed time-point two assessments one week after engaging in CC lessons and activities. With this one-week delay from completing CC lessons and activities to completing the PCK<sub>2</sub> assessment, participants in the experimental group may not have remembered the functions of various code blocks, and thus performed more poorly than if they had completed the assessment closer to the time when they finished CC lessons and activities, as the control group did.

Regardless of the differences in the total number of instructional and experiential hours between the control and experimental groups, the total duration of the course (i.e., the treatment dosage) may not have been enough to affect statistical or meaningful change in learning, or the selected assessments may not have been sensitive enough to change within the relatively short test-retest time interval. Relatedly, instructional hours were not necessarily instructional, but rather, a mix of independent or guided exploration in Scratch, supplemented by occasional one-to-one assistance and whole-group instruction. Some children relied more heavily on the Scratch workbooks than others, while others preferred to learn how to use the Scratch environment on their own by more of a trial-and-error, or haphazard style. Still, other children tended to experience

frustration with lessons and activities, and when instructors were not immediately able to provide assistance, these children often engaged in other computer activities, i.e., watching YouTube videos or playing non-Scratch games on the web. The actual amount of Scratch-engaged time, thus, varied considerably across participants, and there was no direct way to measure the amount of time each participant spent actually manipulating code blocks to complete lessons and create projects. In an attempt to reduce non-Scratch computer activities, the instructors instituted class wide rules aimed at relegating computer usage to only Scratch based activities with repeated noncompliance resulting in a locked computer screen; however, these rules were met with great resistance, and were difficult to enforce without major disruption in classroom activities. Subsequently, for the sake of the children who were engaging appropriately with the CC lessons and activities, instructors chose to allow participants to move to non-Scratch activities after completing one lesson or task. This resulted in some children spending more time in non-Scratch activities than others.

Similar to the aforementioned possible link between length of PCK<sub>1</sub> items, and perceived difficulty of the item, the items comprising the CPS assessments may have been written at too high of a level of linguistic and conceptual complexity for participants to truly grasp the task. This notion was reflected in the large proportion of raters indicating that they thought a participant had misinterpreted or misunderstood items. The CPS assessments were developed using Kim, Chung, and Yu's (2013) synthetic creative problem solving test, which was a study that included a group of South Korean students classified as gifted, in addition to those who were not classified as gifted. Although the language of items comprising both the CPS<sub>1</sub> and CPS<sub>2</sub> was adjusted to reflect linguistic

and cultural differences, it did not undergo a validation procedure to ensure youth in the urban mid-Atlantic region of the United States were able to comprehend the prompts.

Furthermore, the lack of agreement, and in fact, evidence for more disagreement among raters, suggests that raters did not view the originality and usefulness dimensions of creativity in the same way for the same children. With more qualified raters undergoing more comprehensive training on the consensual assessment through rating a variety of mock participant responses designed to represent predetermined degrees of originality and usefulness, there may have been a higher degree of inter-rater reliability. Lastly, as it relates to the CPS assessments, although each rater was provided with participant responses to both CPS assessments in different orders, and one of the three raters was provided with a reversed presentation of item prompts, the raters' assessment packets were organized by CPS<sub>1</sub> and CPS<sub>2</sub>. Therefore, raters may have become fatigued, or changed their internal criteria of usefulness or originality throughout the course of seeing participant responses depending on which of the CPS assessments they looked at first. If all participant responses to each CPS assessment had been shuffled into an overarching response packet with a random order of CPS<sub>1</sub> and CPS<sub>2</sub> assessments, any order effects would have been reduced.

Another limiting factor was the measure chosen to assess mathematical problem solving ability. The KTEA-3 Math Concepts and Applications subtest focused on the conceptual component of mathematics rather than calculation skills, or operational component of mathematics. While it was hypothesized that children engaging in computer programming activities would acquire more mathematic conceptual knowledge, success with achieving a desired goal in Scratch was often paired with specific math

calculation skills; therefore, a direct assessment of calculation skills may have been a more appropriate measure of mathematical thinking as it relates to realizing desired goals of Scratch projects.

Finally, the WJ-IV CF test was artificially split into a pre and post-test, making an already difficult test to administer even more difficult. Despite efforts made to ensure that research assistants demonstrated administrative competency during individual training sessions, there were still three administration errors on the pre-test (WJ-IV CF<sub>1</sub>), and on all three occasions, assessors discontinued the test, mistakenly believing that the youth had reached the discontinue criterion at one of two decision points. All three errors were also all for participants who were in the experimental group, resulting in a possible underestimation of these participants' ability to apply inductive reasoning to solve problems; however, these participants' performance on the WJ-IV CF<sub>2</sub>, where there were no administration errors, was less than or equal to their performance on the WJ-IV CF<sub>1</sub>, so it may have been possible that they would not have obtained any additional points even if administration errors had not occurred on the pre-test.

### Conclusion

The study of computational thinking and its component cognitive processes is ripe for expansion and clarification. Ongoing efforts to further decompose and delineate the development of CT skills will be important to better inform both general and CS specific educators on developmental expectations across age bands, and incorporate effective instructional strategies and curricular content. From the results of the current study, it is clear that informal learning environments such as a summer “camp” setting present a number of challenges related to attendance and motivation. It was also clear that the

Creative Computing Curriculum activities and lessons needed to be altered to bolster participants' engagement with computer programming activities. The lack of differences between groups across measures related to problem solving and creativity may have been related to the somewhat low treatment dosage and short time span of the study (roughly 25 hours over two weeks), in addition to the more introductory level of engagement with curricular material. Lastly, the measurement of CT skills and processes was difficult to interpret given the selected measures for the study, and future research or instruction should consider additional efforts to more thoroughly explore what is going on in the minds of children as the attempt to engage in computational thinking practices through differing assessment methods.

One area that may be of particular importance for future research and computer science education is the language used by instructors, and embedded within lessons and activities to describe CT skills and processes. The way in which instructors describe what to do as it relates to key CT concepts is important to consider, and has been suggested as critical for educators to operationally standardize in order for student clarity and understanding (Waite, Curzon, Marsh, Sentence, & Hadwen-Bennet, 2018). The language embedded within the categories and names of code in the Scratch programming environment are designed to be intuitive, but children often needed additional, repeated explanations of the function of various code blocks throughout lessons and activities of the CC curriculum. CS educators should be cognizant and conscious of the type of language they use to describe computer programs, understanding that the conceptualization of key computational processes is highly dependent on the language of instruction.

While there is much work to be done, and despite the many challenges and obstacles to overcome, understanding the implications for incorporating CT activities and lessons within the general education curriculum will only gain momentum as computer science grows in its importance for shaping the modern world.

## REFERENCES CITED

- Aho, A. V. (2012). Computation and computational thinking. *Computer Journal*, 55, 832–835.
- Akcaoglu, M., & Koehler, M. J. (2014). Cognitive outcomes from the Game-Design and Learning (GDL) after-school program. *Computers & Education*, 75, 72-81.
- Allsop, Y. (2015). A reflective study into children's cognition when making computer games. *British Journal of Educational Technology*. Advance online publication. <http://dx.doi.org/10.1111/bjet.12251>
- Almeida, L. S., Prieto, L. P., Ferrando, M., Oliveira, E., & Ferrándiz, C. (2008). Torrance Test of Creative Thinking: The question of its construct validity. *Thinking Skills and Creativity*, 3, 53-58.
- Amabile, T. M. (1982). Social psychology of creativity: A consensual assessment technique. *Journal of Personality and Social Psychology*, 43, 997–1013.
- Au, W., & Leung, J. (1991). Problem-solving, instructional-methods and logo programming. *Journal of Educational Computing Research*, 7(4), 455-467.
- Bebras. (2018). Retrieved from <http://bebras.org/?q=goodtask>
- Bebras Computing Challenge. (2018). Retrieved from <http://www.bebрасchallenge.org/index.php>
- Baer, J., & McKool, S. S. (2009). Assessing Creativity Using the Consensual Assessment Technique. In C. Schreiner, *Handbook of assessment technologies, methods, and applications in higher education*. Hershey, Pennsylvania: IGI Global.



- Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads*, 2, 48–54.
- Battista, M. T., & Clements, D. H. (1986). The Effects of Logo and CAI problem-solving Environments on Problem-solving Abilities and Mathematics Achievement. *Computers in Human Behavior*, 2(3), 183-193.
- Baytak, A. & Land, S. (2011). An investigation of the artifacts and process of constructing computer games about environmental science in a fifth grade classroom. *Educational Technology Research & Development*, 59(6), 765-782.
- Bell, T., Andreae, P., & Robins, A. (2012). Computer science in NZ high schools: The first year of the new standards. In *Proceedings of the 43rd ACM technical symposium on computer science education*. Raleigh, NC.
- Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Annual American Educational Research Association Meeting*, Vancouver, BC, Canada.
- Brennan, K. Balch, C., & Chung, M. (2014). Creative Computing: A design-based introduction to computational thinking. *Harvard Graduate School of Education*. Retrieved from: <http://scratched.gse.harvard.edu/guide/>.
- Bruner, J. S. (1960). *The Process of Education*. Cambridge, MA: Harvard University Press.
- Bryant R. E., Katz R. H., & Lazowska E. D. (2008). Big-Data Computing: Creating revolutionary breakthroughs in commerce, science, and society: A white paper

- prepared for the Computing Community Consortium committee of the Computing Research Association. <http://cra.org/ccc/resources/ccc-led-whitepapers/>
- Calao, L. A., Moreno-León, J., Correa, H. E., & Robles, G. (2015). Developing mathematical thinking with scratch an experiment with 6th grade students. Paper presented at the, 9307 17-27. doi:10.1007/978-3-319-24258-3\_2
- Caspersen, M. E., & Nowack, P. (2013). Computational thinking and practice: A generic approach to computing in Danish high schools. In *Proceedings of the fifteenth Australasian computing education conference* (Vol. 136, pp. 137–143). Sydney: Australian Computer Society.
- Carnevale, A. P., Smith, N., & Melton, M. (2011). *STEM: Science, Technology, Engineering, Mathematics*. Retrieved from Georgetown University Center on Education and the Workforce <http://cew.georgetown.edu/stem>.
- Castells, M. (1996). *The rise of the network society*. Cambridge, Mass: Blackwell Publishers
- Chambless, D. L., & Hollon, S. D. (1998). Defining Empirically Supported Therapies. *Journal of Consulting and Clinical Psychology*, 66(1), 7-18.
- Cho, S., Jang, Y., Jung, T., Lim, H., & Park, I. (2002). *Development of creative problem solving test II (ED)*. Seoul: KEDI (in Korean).
- Choi, J., An, S., & Lee, Y. (2015). Computing Education in Korea—Current Issues and Endeavors. *ACM Transactions on Computing Education*, 15(2), 8:2-8:22.
- Clements, D. H. (1986). Effects of logo and CAI environments on cognition and creativity. *Journal of Educational Psychology*, 78(4), 309-318. doi:10.1037/0022-0663.78.4.309

- Clements, D. H. (1990). Metacomponential development in a logo programming environment. *Journal of Educational Psychology*, 82(1), 141-149.  
doi:10.1037/0022-0663.82.1.141
- Clements, D., & Nastasi, B. (1999). Metacognition, learning, and educational computer environments. *Information Technology in Childhood Education Annual*, (1), 5–38.
- Clements, D. H., & Sarama, J. (1995). Design of a Logo Environment for Elementary Geometry. *Journal of Mathematical Behavior*, 14, (4), 381-398.
- Code.org (2018). CS Fundamentals Standards Alignment. Retrieved from <https://curriculum.code.org/csf-1718/standards/>
- Cohen, R. (1987). Implementing Logo in the grade two classroom: Acquisition of Basic programming concepts. *Journal of Computer-Based Instruction*, 14(4), 124-132.
- College Board. (2015). AP course audit. Retrieved from <https://apcourseaudit.epiconline.org/ledger/search.php>
- Computer Science Education Act, H. R. 5929, 111<sup>th</sup> Cong. (2010).
- Computer Science Teachers Association, Standards Task Force (2011). *Report on the CSTA K-12 Computer Science Standards*. Retrieved from [http://csta.acm.org/Curriculum/sub/CurrFiles/CSTA\\_K-12\\_CSS.pdf](http://csta.acm.org/Curriculum/sub/CurrFiles/CSTA_K-12_CSS.pdf)
- Computer Science Teachers Association (2017). *CSTA K-12 Computer Science Standards*. Revised 2017. Retrieved from <http://www.csteachers.org/standards>
- Computing at Schools Working Group (2012). A curriculum framework for Computer Science and Information Technology.

- Cuban, L. (2001). *Oversold and underused: Computers in the classroom*. Boston: Harvard University Press.
- Dagienė, V. (2006). Information technology contests – introduction to computer science in a attractive way. *Informatics in Education*, 5(1), 37–46.
- Dagienė, V., & Stupurienė, G. (2016). Bebras – a Sustainable Community Building Model for the Concept Based Learning of Informatics and Computational Thinking. *Informatics in Education*, 15(1), 25-44.
- DeCorte, E. (1992). On the learning and teaching of problem-solving skills in mathematics and logo programming. *Applied Psychology-an International Review-Psychologie Appliquee-Revue Internationale*, 41(4), 317-331.  
doi:10.1111/j.1464-0597.1992.tb00709.x
- Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers and Education*, 58(1), 240-249.
- diSessa, A. A. (2000). *Changing minds: Computers, learning, and literacy*. The MIT Press; Cambridge, MA.
- Dynarski, M., James-Burdumy, S., Moore, M., Rosenberg, L., Deke, J., & Mansfield, W. (2004). *When School Stay Open Late: The National Evaluation of the 21<sup>st</sup> Century Community Learning Centers Program: New Findings*. U.S. Department of Education, National Center for Education Evaluation and Regional Assistance. Washington, DC: U.S. Government Printing Office.

- Emihovich, C., & Miller, G. E. (1988). Effects of Logo and CAI on Black First Graders' Achievement, Reflectivity, and Self-esteem. *The Elementary School Journal*, 88(5), 473-487. doi:10.1086/461551
- Farley, F. (1986, May). The Big T in Personality. *Psychology Today*, 44-52.
- Fay, A., & Mayer, R. (1994). Benefits of Teaching Design Skills Before Teaching Logo Computer-programming - Evidence for Syntax-independent Learning. *Journal of Educational Computing Research*, 11(3), 187-210.
- Federation of American Scientists. (2006). Harnessing the power of video games for learning. *Summit on Educational Games*. Retrieved from [http://informal.science.org/research/ic-000-000-009-386/Harnessing\\_the\\_Power\\_of\\_Video\\_Games\\_for\\_Learning](http://informal.science.org/research/ic-000-000-009-386/Harnessing_the_Power_of_Video_Games_for_Learning).
- Fessakis, G., Gouli, E., and Mavroudi, E., (2013). Problem solving by 5-6 years old kindergarten children in a computer programming environment: A case study. *Computers and Education*, 63, 87-97.
- Feurzeig, W. (1986). Algebra Slaves and Agents in a Logo-based Mathematics Curriculum. *Instructional Science*, 14, 229-254.
- Fields, D. A., Quirke, L., Amely, J., & Maughan, J. (2016). Combining Big Data and Thick Data Analyses for Understanding Youth Learning Trajectories in a Summer Coding Camp. In *Proceedings of the 47<sup>th</sup> ACM Technical Symposium on Computing Science Education (SIGSCE '16)* (pp. 150-155). New York, NY: ACM.
- Fields, D., Quirke, L., Horton, T., Maughan, J., Velasquez, X., Amely, J., Pantic, K. (2016). Working toward Equity in a Constructionist Scratch Camp: Lessons

- Learned in Applying a Studio Design Model. In A. Sipitakiat & N. Tutiyaphuengprasert (Eds.), *Proceedings of Constructionism in Action, Bangkok, Thailand* (pp. 291-298).
- Flanagan, D. P., Ortiz, S. O., and Alfonso, V. C. (2013). *Essentials in Cross-Battery Assessment*.
- Flavell, J. H. (1976). Metacognition and cognitive monitoring: A new area of cognitive-development inquiry. *American Psychologist*, 34(10), 906-911.
- Fleischman, H.L., Hopstock, P.J., Pelczar, M.P., & Shelley, B.E. (2010). Highlights From PISA 2009: Performance of U.S. 15-YearOld Students in Reading, Mathematics, and Science Literacy in an International Context (NCES 2011-004). U.S. Department of Education, National Center for Education Statistics. Washington, DC: U.S. Government Printing Office.
- Gallup (2014). *Searching for Computer Science: Access and Barriers in U.S. K-12 Education*. Retrieved from <http://csedu.gallup.com>.
- Gee, J. (2003). *What video games have to teach us about learning and literacy*. New York: Palgrave Macmillan.
- Gerber, B. L., Cavallo, A. M. L., & Marek, E. A. (2001). Relationships among informal learning environments, teaching procedures and scientific reasoning ability. *International Journal of Science Education*, 23(5), 535-549.
- Geva, E., & Cohen, R. (1987). Understanding Turn Commands in Logo: A Cognitive Perspective. *Instructional Science*, 16(4), 337-350. doi:10.1007/BF00117751

- Gibbons, P. (1995). A Cognitive Processing Account of Individual Differences in Novice Logo Programmers' Conceptualisation and use of Recursion. *Journal of Educational Computing Research*, 13(3), 211-226.
- Grover, S., & Pea, R. (2013). Computational Thinking in K-12: A Review of the State of the Field. *Educational Researcher*, 42(1), 38-43.
- Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, 25 (2), 1-39.
- Guilford, J. P. (1967). *The nature of human intelligence*. New York: McGraw-Hill.
- Hayes, E. R., & Games, I. A. (2008). Making Computer Games and Design Thinking: A Review of Current Software and Strategies. *Games and Culture*, 3(3-4), 309-332.
- Howard, J., Watson, J., & Allen, J. (1993). Cognitive-style and the Selection of Logo Problem-solving Strategies by Young Black-children. *Journal of Educational Computing Research*, 9(3), 339-354.
- Hoyles, C., Sutherland, R., & Evans, J. (1986). Using logo in the mathematics classroom. what are the implications of pupil devised goals? *Computers & Education*, 10(1), 61-71. doi:10.1016/0360-1315(86)90053-9
- Hwang, G., Hung, C., & Chen, N. (2014). Improving learning achievements, motivations, and problem-solving skills through a peer assessment-based game development approach. *Educational Technology Research and Development*, 62(2), 129-145.
- Intergovernmental Panel on Climate Change. (2013). Summary for policymakers. In T. F. Stocker, D. Qin, G.-K., Plattner, M. Tignor, S. K. Allen, J. Boschung, A. Nauels, Y., Xia, V. Bex, P. M. Midgley (Eds.), *Climate Change 2013: The physical*

- science basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change.* Cambridge, UK: Cambridge University Press. Retrieved from <http://www.ipcc.ch/report/ar5/wg1>
- Ioannidou, A., Repenning, A., Webb, D. C. (2009). AgentCubes: Incremental 3D end-user development. *Journal of Visual Languages and Computing*, 20, 236-251.
- Ito, M. (2008). Education vs. Entertainment: A Cultural History of Children's Software. In Salen, K. (Ed.), *The Ecology of Games: Connecting Youth, Games, and Learning*. (pp. 89-116). Cambridge, MA: The MIT Press. doi: 10.1162/dmal.9780262693646.089
- Izu, C., Mirolo, C., Settle, A., Mannila, L., & Stupurienè. (2017). Exploring Bebras Tasks Content and Performance: A Multinational Study. *Informatics in Education*, 16(1), 39-59, doi: 10.15388/infedu.2017.03
- Jonassen, D. H. (2004). *Learning to solve problems: An instructional design guide*. San Francisco, CA: Pfeiffer.
- Kafai, Y. B. (1995). *Minds in play: Computer game design as a context for children's learning*. Mahwah, NJ: Lawrence Erlbaum.
- Kafai, Y. B., & Burke, Q. (2013, September). Computer programming goes back to school. *Kappan*, 95(1), 61-65.
- Kafai, Y. B., & Burke, Q. (2014). *Connected Code: Why Children Need to Learn Programming*. Cambridge, MA: The MIT Press.
- Kafai, Y. B., & Burke, Q. (2016). Constructionist Gaming: Understanding the Benefits of Making Games for Learning. *Educational Psychologist*, 50(4), 313-332, doi: 10.1080/00461520.2015.1124022



- Kafai, Y. B., Peppler, K. A., & Chiu, G. M. (2007). High tech programmers in low-income communities: Creating a computer culture in a community technology center. In *Communities and Technologies 2007: Proceedings of the Third Communities and Technoogies Conference* (pp. 545-563) doi:10.1007/978-1-84628-905-7\_27
- Kafai, Y. B., & Resnick, M. (Eds.) (1994). *Constructionism in practice: Designing, thinking and learning in a digital world*. Mahwah, NJ: Lawrence Erlbaum Associates.
- Kaufman. A. S., & Kaufman, N. L. (2014). *Kaufman test of educational achievement, third edition*. Bloomington, MN: NCS Pearson.
- Kaufman, J.C., & Baer, J. (2012). Beyond New and Appropriate: Who Decides What is Creative? *Creativity Research Journal*, 24(1), 83-91. doi: 10.1080/10400419.2012.649237
- Kelleher, C. & Pausch, R. (2003). Lowering the barriers to programming: a taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys*, 37(2), 88-137.
- Khalili, N., Sheridan, K., Williams, A., Clark, K., & Stegman, M. (2011). Students designing video games about immunology: Insights for science learning. *Computers in the Schools*, 28(3), 228-240. doi:10.1080/07380569.2011.594988
- Khasawneh, A. A. (2009). Assessing Logo Programming among Jordanian Seventh Grade Students through Turtle Geometry. *International Journal of Mathematical*

- Education in Science and Technology*, 40(5), 619-639.  
doi:10.1080/00207390902912845
- Kim, S., Chung, K., & Yu, H. (2013). Enhancing digital fluency through a training program for creative problem solving using computer programming. *The Journal of Creative Behavior*, 47(3), 171-199.
- Kisiel, J. (2005). Understanding elementary teacher motivations for science fieldtrips. *Science Education*, 89(6), 936-955.
- Kurland, D. M., & Pea, R. D. (1985). Children's Mental Models of Recursive Logo Programs. *Journal of Educational Computing Research*, 1(2), 235-243.
- Leandro, S.A., Lola, P.P., Mercedes, F., Emma, O., & Carmen, F. (2008). Torrance test of creative thinking: The question of its construct validity. *Thinking Skills and Creativity*, 3, 53-58.
- Li, Q. (2010). Digital game building: learning in a participatory culture. *Educational Research*, 52(4), 427-443.
- Liao, Y-K. C., & Bright, G. W. (1991). Effects of Computer Programming on Cognitive Outcomes: A Meta-Analysis. *Journal of Educational Computing Research*, 7(3), 251-268.
- Lin, J. M., Li, Y., Ho, R., & Li, C. (2007). Effects of guided collaboration on sixth graders' performance in logo programming. Paper presented at the T1B-11-T1B-16. doi:10.1109/FIE.2007.4418172
- Littlefield, J., Delclos, V. R., Bransford, J. D., Clayton, K. N., & Franks, J. J. (1989). Some Prerequisites for Teaching Thinking: Methodological Issues in the Study of

- LOGO Programming. *Cognition and Instruction*, 6(4), 331-366.  
doi:10.1207/s1532690xci0604\_4
- Logo History. (2015). Retrieved from [http://el.media.mit.edu/logo-foundation/what\\_is\\_logo/history.html](http://el.media.mit.edu/logo-foundation/what_is_logo/history.html)
- Lohr, S. (2012, February 11). The Age of Big Data. *The New York Times*. Retrieved from [http://www.nytimes.com/2012/02/12/sunday-review/big-datas-impact-in-the-world.html?\\_r=0](http://www.nytimes.com/2012/02/12/sunday-review/big-datas-impact-in-the-world.html?_r=0)
- Lucas, K.B. (2000). One teacher's agenda for a class visit to an interactive science center. *Science Education*, 84, 545-544.
- Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, 41, 51-61.
- Maloney, J., Peppler, K., Kafai, Y. B., Resnick, M., Rusk, N. (March, 2008).  
Programming by Choice: Urban Youth Learning Programming with Scratch.  
Proceedings published by the ACM Special Interest Group on Computer Science Education, Portland, OR.
- Mather, N., and Wendling, B. J. (2014). Examiner's Manual. *Woodcock-Johnson IV Tests of Cognitive Abilities*. Rolling Meadows, IL: Riverside.
- Mayer, R. E. (1977). *Thinking and problem solving: An introduction to human cognition and learning*. Glenview, IL: Scott, Foresman.
- Mayer, R. E. (1999). Fifty years of creativity research. In R. J. Sternberg (Ed.),  
Handbook of human creativity (pp. 449-460). New York, NY: Cambridge University Press.

- Mayer, R. E., & Fay, A. L. (1987). A Chain of Cognitive Changes with Learning to Program in Logo. *Journal of Educational Psychology*, 79 (3), 269-279.
- National Research Council. (2010). *Exploring the intersection of science education and 21st century skills: A workshop summary*. M. Hilton (Ed.). Washington, DC: National Academy Press.
- National Research Council. (2012). *A Framework for K-12 Science Education: Practices, Crosscutting Concepts, and Core Ideas*. Quinn, H., Schweingruber, H., and Keller, T. (Ed.s). Washington, DC: National Academy Press.
- National Science Foundation. (2005). *National Science Board 2020 Vision for the National Science Foundation*. (NSB Report-05-142). Retrieved from <http://www.nsf.gov/publications/orderpub.jsp>
- National Science Foundation. (2009). A Week to Focus on Computer Science Education [Press Release]. Retrieved from [http://www.nsf.gov/news/news\\_summ.jsp?cntn\\_id=116059](http://www.nsf.gov/news/news_summ.jsp?cntn_id=116059)
- Noss, R. (1986). Constructing a conceptual framework for elementary algebra through logo programming. *Educational Studies in Mathematics*, 17(4), 335-357.  
doi:10.1007/BF00311324
- Noss, R. (1987). Children's learning of geometrical concepts through Logo. *Journal for Research in Mathematics Education*, 18(5), 343-362.
- Olive, J. (1991). Logo programming and geometric understanding: An in-depth study. *Journal for Research in Mathematics Education*, 22(2), 90-111.
- Owston, R., Wideman, H., Ronda, N. S., & Brown, C. (2009). Computer game development as a literacy activity. *Computers & Education*, 53, 977-989.

- Palumbo, D. B. (1990). Programming Language/Problem-Solving Research: A Review of Relevant Issues. *Review of Educational Research*, 60(1), 65-89.
- Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books Inc.: New York, NY.
- Papert, S. (1991). Situating constructionism. In Harel and Papert (Eds.), *Constructionism* (pp. 1-12). Norwood, NJ: Ablex.
- Pantic, K. Fields, D. A., & Quirke, L. (2016). Studying situated learning in a constructionist programming camp: A multimethod microgenetic analysis of one girl's learning pathway. *Proceedings of the 15<sup>th</sup> International Conference on Interaction Design and Children, Manchester, United Kingdom*, 428-439. doi: 10.1145/2930674.2930725.
- Pardamean, B., & Evelyn, T. S. (2014). Enhancement of creativity through logo programming. *American Journal of Applied Sciences*, 11(4), 528-533. doi:10.3844/ajassp.2014.528.533
- Pea, R. D., & Kurland, D. M. (1984). On the Cognitive Effects of Learning Computer Programming. *New Ideas in Psychology*, 2(2), 137-168.
- Piaget, J. (1936). *Origins of Intelligence in the Child*. London, UK: Routledge & Kegan Paul.
- Piaget, J. (1951). *Play, dreams, and imitation in childhood*. New York, NY: Norton.
- Piaget, J. (1962). *The language and thought of the child* (3rd, rev. and enl. ed.). New York; London;: Routledge. (Original work published 1926)

- Pinkston, G. (2015). Forward 50, Teaching Coding Ages 4-12: Programming in the Elementary School. *5<sup>th</sup> Annual International Conference on Education & e-Learning (EeL 2015)*. doi: 10.5176/2251-1814\_EeL15.11
- Poulin-Dubois, D., McGilly, C. A., & Shultz, T. R. (1989). Psychology of computer use: X. effect of learning logo on children's problem-solving skills. *Psychological Reports, 64*(3), 1327-1337. doi:10.2466/pr0.1989.64.3c.1327
- Polya, G. (1957). *How to solve it*. Garden City, NY: Doubleday/Anchor.
- Rasch, G. (1960). *Probabilistic models for some intelligence and attainment tests*. Copenhagen, Denmark: Danish Institute for Educational Research.
- Resnick, M., Kafai, Y., & Maeda, J. (2003). ITR: A Networked, Media-Rich Programming Environment to Enhance Technological Fluency at After-School Centers. Proposal [funded] to the National Science Foundation, Washington, DC.
- Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for All. *Communications of the ACM, 52*(11), 60-67.
- Román-González, M., Pérez-González, J. C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in Human Behavior, 72*, 678-691.
- Royal Society. (2012). Shut down or restart: The way forward for computing in UK schools. Retrieved from <http://royalsociety.org/education/policy/computin-in-schools/report/>
- Rushkoff, D. (2011). *Program or Be Programmed? Ten Commands for a Digital Age*. New York: Soft Skull Press.

- Sarshar, M. (2017). *Explorations in Type-T: Mindset, Flourishing, Psychological Entitlement, Creativity, and Stress* (Doctoral dissertation). Retrieved from ProQuest Dissertations Publishing. (10683365)
- Schanzer, E., Fisler, K., & Krishnamurthi, S. (2013, October). *Bootstrap: Going beyond programing in after-school computer science*. Presented at SPLASH-E, Education track of the OOPSLA/SPLASH conference, Indianapolis, IN.
- Schanzer, E., Fisler, K., Krishnamurthi, S., & Felleisen, M. (2015). Transferring skills at solving word problems from computing to algebra through Bootstrap. In *Proceedings of the 46th ACM Technical symposium on computer science education* (pp. 616–621). New York, NY: ACM.
- Schrank, F. A., McGrew, K. S., & Mather, N. (2014). *Woodcock-Johnson IV Tests of Cognitive Abilities*. Rolling Meadows, IL; Riverside.
- Selby, C., Dorling, M. & Woollard, J., (2014). Evidence of assessing computational thinking. Retrieved from:  
<http://eprints.soton.ac.uk/372409/1/372409EvidAssessCT.pdf>
- Shodor. (2016). Retrieved from <http://www.shodor.org/>
- Squire, K. (2006). From content to context: Videogames as designed experience. *Educational Researcher*, 35(8), 19-29.  
 doi:10.3102/0013189X035008019
- STEM Education Act, H. R. 5031, 113<sup>th</sup> Cong. (2014).
- Sternberg, R. J. (1985) *Beyond IQ: A triarchic theory of human intelligence*. London: Cambridge University Press.
- Sternberg, R. J. (1994). *Thinking and problem solving*. San Diego, CA: Academic Press.

- Straw, S., Bamford, S., & Styles, B. (2017). Randomised Controlled Trial and Process Evaluation of Code Clubs. Slough: NFER.
- Suomala, J. (1996). Eight-year-old-pupils' Problem-solving processes within a LOGO Learning Environment. *Scandinavian Journal of Educational Research*, 40(4), 291-309. doi:10.1080/0031383960400402
- Sutherland, R. (1993). Connecting Theory and Practice: Results from the Teaching of Logo. *Educational Studies in Mathematics*, 24(1), 95-113.  
doi:10.1007/BF01273296
- Tai, R., Liu, C. Q., Maltese, A. V., & Fan, X. (2006). Career choice: Enhanced: Planning early for careers in science. *Science*, 312, 1143–1144.
- The College Board (2017). *AP Computer Science Principles, Course and Exam Description*. New York: NY.
- The White House (2016, January 30). Computer Science for All [Web log post]. Retrieved from <https://www.whitehouse.gov/blog/2016/01/30/computer-science-all>
- Torrance, E.P. (1966). *The Torrance Tests of Creative Thinking –Norms-Technical Manual Research Edition – Verbal Tests, Forms A and B – Figural Tests, Forms A and B*. Princeton, NJ: Personnel Press.
- Torrance, E. P. (1969). *Creativity. What research says to the teacher*. Washington, DC: National Education Association.
- Tracz, W. (1979). Computer Programming and the Human Thought Process. *Software – Practice and Experience*, 9, 127-137.
- UK Bebras Computational Thinking Challenge. (2016). *2016 Answers*. Retrieved from



- <http://www.bebbras.uk/uploads/2/1/8/6/21861082/uk-bebbras-2016-answers.pdf>
- US Bureau of Labor Statistics. (2013). *Industry employment and output projections to 2022* (Monthly Labor Review). Retrieved from <http://www.bls.gov/opub/mlr/2013/article/pdf/industry-employment-and-output-projections-to-2022.pdf>
- US Department of Education, National Commission on Excellence in Education (1983). *A Nation at Risk: The Imperative for Educational Reform: A report to the Nation and the Secretary of Education*. Washington, DC: Government Printing Office.
- Valente, J. A. (2003). How logo has contributed to the understanding of the role of informatics in education and its relation to the learning process. *Informatics in Education*, 2(1), 123-138.
- Vygotsky, L. (1978). *Mind in society: The development of higher psychological processes*. M. Cole, V. John-Steiner, S. Scribner, & E. Souberman (Eds.). Cambridge, MA: Harvard University Press.
- Waite, J. L., Curzon, P., Marsh, W., Sentence, S., & Hadwen-Bennett, A. (2018). Abstraction in action: K-5 teachers' uses of levels of abstraction, particularly the design level, in teaching programming. *International Journal of Computer Science Education in Schools*, 2(1),
- Werner, L., Denner, L., & Campe, S. (2014). Children Programming Games: A Strategy for Measuring Computational Learning. *ACM Transactions on Computing Education*, 14(4),
- Werner, L., Denner, L., Campe, S., & Kawamoto, D. C. (2012). The Fairy Performance Assessment: Measuring Computational Thinking in Middle School. In

- Proceedings of the 44<sup>th</sup> ACM technical symposium on computer science education (SIGSCE '12)* (pp. 215-220). New York, NY: ACM.
- Werner, L., Hanks, B., & McDowell, C. (2004). Pair programming helps female computer science students. *ACM Journal of Educational Resources in Computing*, 4(1).
- Wing, J. (2006). Computational Thinking. *Communications of the ACM*, 49(3), 33-35.
- Wing, J. (2008). Computational Thinking and thinking about computing. *Philosophical Transactions of the Royal Society*, 366, 3717-3725.
- Wing, J. (2011). Research notebook: Computational thinking—What and why? *The Link Magazine*, Spring. Carnegie Mellon University, Pittsburgh. Retrieved from <http://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why>
- Wilson, C., Sudol, A. L., Stephenson, C., & Stehlik, M. (2010). *Running on Empty: The Failure to Teach K-12 Computer Science in the Digital Age*. Retrieved from <http://csta.acm.org/runningonempty/fullreport.pdf>.
- Wilson, D., Mundy-Castle, A., & Sibanda, P. (1991). Cognitive Effects of LOGO and Computer-aided Instruction among Black and White Zimbabwean Primary School Girls. *The Journal of Social Psychology*, 131(1), 107.
- Wolz, U., Hallberg, C., & Taylor, B. (March, 2011). *Scrape: A tool for visualizing the code of Scratch programs*. Poster presented at the 42nd ACM Technical Symposium on Computer Science Education, Dallas, TX.
- Wright, B.D., and Stone, M.H. (1979). *Best test design*. Chicago, IL; MESA Press.

- Wu, M. L., & Richards, K. (2011). Facilitating computational thinking through game design. In M. Chang, W. Y. Hwang, M. P. Chen, & W. Müller (Eds.), *Edutainment Technologies. Educational Games and Virtual Reality/Augmented Reality Applications* (pp. 220-227). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-23456-9\_39
- Yagunova, E., Podznyakov, S., Ryzhova, N., Razumovskaia, E., Korovkin, N. (2015). Tasks classification and age differences in task perception: Case study of international on-line competition “Beaver”. In: *Proc. of the 8th ISSEP Conf.* Univ. of Ljubljana, pp. 33–43.
- Yelland, N. J. (1995). Encouraging young children's thinking skills with logo. *Childhood Education*, 71(3), 152-155. doi:10.1080/00094056.1995.10521831
- Yoo, S. W., Yeum, Y. C., Kim, Y., Cha, S. E., Kim, J. H., Jang, H. S., Choi, S. K., Lee, H. C., Kwon, D. Y., Han, H. S., Shin, E. M., Song, J. S., Park, J. E., and Lee, W. G. (2006). Development of an integrated informatics curriculum for K-12 in Korea. In *Informatics Education: The Bridge between Using and Understanding Computers*, Roland Mittermeir (Ed.), Lecture Notes in Computer Science, Vol. 4226. Springer, 199–208. DOI: <http://dx.doi.org/10.1007/11915355-19>.
- Zur-Bargury, I. (2012). A new curriculum for junior-high in computer science. In *Proceedings of the 17th ACM annual conference on innovation and technology in computer science education*. (pp. 204–208). New York, NY: ACM.

## APPENDIX A

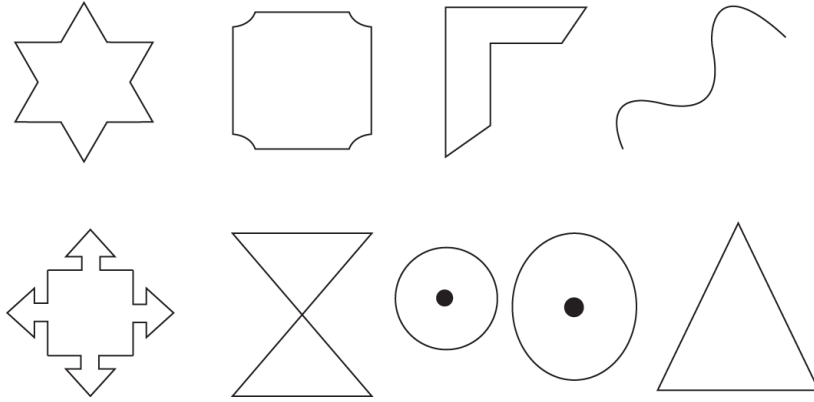
### CREATIVE PROBLEM SOLVING PRE-TEST

#### Prompt 1

CPS Pre-test

Much of the worlds' oceans have yet to be explored. To explore the ocean, high-tech machines and equipment are needed. In the deep parts of the ocean, there are mysterious animals, underwater volcanoes, and resources that people can use. There may even be things that no one has ever seen before!

Invent something to explore the deep ocean. This invention must be able to work in the deepest parts of the ocean. Draw your invention using the given shapes. Name and explain your invention by describing the usefulness of the invention in detail. You can use each shape more than once and shapes can be combined.



**Prompt 2**

CPS Pre-test

Countries across the world have realized that they need to conserve energy resources to protect the environment. Bicycles are good because they do not use up energy resources or pollute the environment. What would you do to improve the disadvantages of the current bike? Write as much as possible about an idea to upgrade anything about the bike that is inconvenient.

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

**Prompt 3**

CPS Pre-test

When baseball was created, it was different than it is now. Today, baseball combines a ball with a wooden bat. Come up with a creative invention by combining two unrelated things or objects. Put a name to your invention and write about why you decided to create the invention.

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

**Prompt 4**

CPS Pre-test

An idea sketch is used to show your thoughts through pictures. The pictures could include comics, scribbles, tables, maps, and simple words. Imagine that you're with someone who doesn't know our language. You have to explain the words below to communicate with him or her. Using an idea sketch, help this person understand as many words of our language as possible.

Word	Pictures	Word	Pictures
pollution		color	
sound		active	
sour		mixture	
wrong		life-cycle	
light		coal	

## APPENDIX B

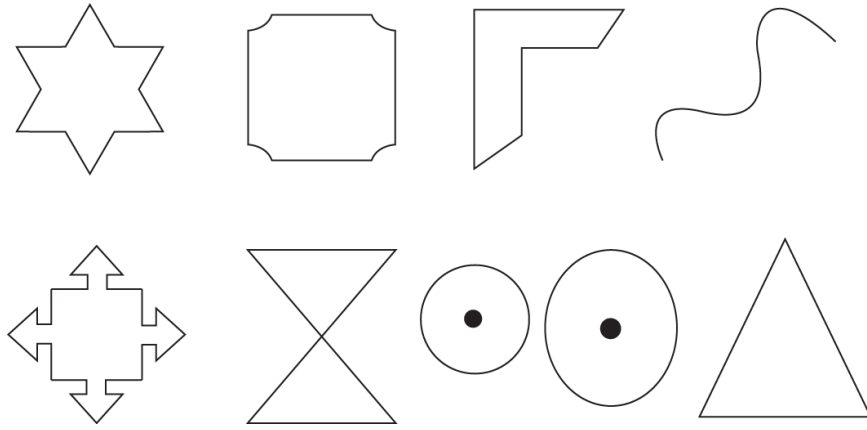
### CREATIVE PROBLEM SOLVING POST-TEST

#### Prompt 1

CPS Post-test

The mysteries of outer space are yet to be discovered, and our ability to understand what may lie beyond our own planet is becoming easier as science and technology advance. To explore outer space, scientists and engineers have to come up with new methods and machines to handle the harsh environment of deep space.

Invent something to explore outer space. The invention must be able to work in the deepest, darkest parts of space, and also on different planets that may have very different environmental conditions than Earth. Draw your invention using the given shapes. Name and explain your invention by describing the usefulness of the invention in detail. You can use each shape more than once, and shapes can be combined.





**Prompt 2**

CPS Post-test

Brooms make it easier to do housework such as cleaning the floor. Although brooms have been around for hundreds of years, the design of the broom has not changed much. What would you do to improve on the disadvantages of the current broom? Write as much as possible about an idea to upgrade anything about the broom that is inconvenient.

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

**Prompt 3**

The first people who caught fish used much more simple equipment than today’s rods and reels. The first fishing pole was probably a stick with rope and a bone hook tied to it. Come up with your own creative invention by combining two unrelated things or objects. Put a name to your invention and write about why you decided to create the invention.

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

**Prompt 4**

An idea sketch is used to show your thoughts through pictures. The pictures could include comics, scribbles, tables, maps, and simple words. Imagine that you're with someone who doesn't know our language. You have to explain the words below to communicate with him or her. Using an idea sketch, help this person understand as many words of our language as possible.

Word	Pictures	Word	Pictures
energy		weight	
smell		lazy	
sweet		machine	
correct		recycle	
dark		plastic	

## APPENDIX C

### TYPE T QUESTIONNAIRE

Read the statements below and circle how much they fit with who you are as a person.

1. I like to do exciting things	NEVER	A LITTLE BIT	A LOT
2. I enjoy taking chances	NEVER	A LITTLE BIT	A LOT
3. I like people who are really different from me	NEVER	A LITTLE BIT	A LOT
4. I like to make up my own mind	NEVER	A LITTLE BIT	A LOT
5. Someday I would like to drive a race car in a fast race	NEVER	A LITTLE BIT	A LOT
6. I like thinking about really unusual things	NEVER	A LITTLE BIT	A LOT
7. I like to do things real fast without thinking too much about it	NEVER	A LITTLE BIT	A LOT
8. I enjoy trying new and different food that I've never had before	NEVER	A LITTLE BIT	A LOT

© Frank Farley 2017

## APPENDIX D

### USA BEBRAS CHALLENGE 2016 QUESTIONS AND ANSWERS

#### Mazes

Castors:  
Benjamins: A  
Cadets:

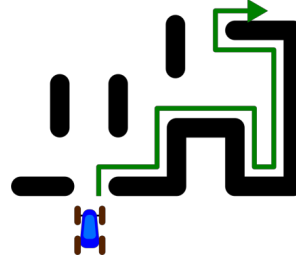
Juniors:  
Seniors:



A robotic car uses a simple rule to drive through a maze:

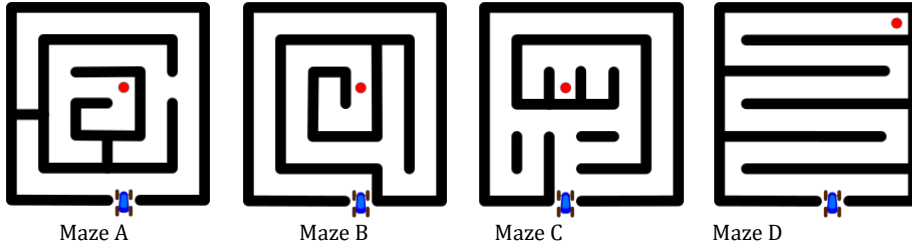
Turn right whenever possible.

The picture on the right gives an example of how the robot would drive through a maze.



#### Question:

In how many of the following mazes will the car reach the red dot if it uses this system?



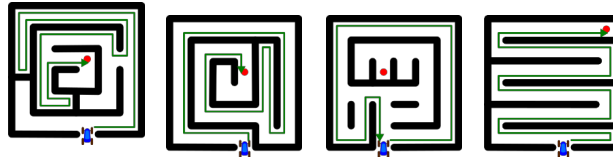
Choose from: 0 1 2 3 or 4

#### Answer:

Maze C

#### Explanation:

In the pictures below, the green arrow indicates the path taken by the car. In Maze C the whole center part of the maze is not visited and the red dot is not reached. In all other cases the red dot is reached.



#### It's Computational Thinking:

CT Skills - Algorithmic Thinking (AL)

CS Domain - Algorithms and programming

The method which is used by our car, is called the wall follower. It is one of the simpler techniques (algorithms) to solve a maze for which you do not know the layout in advance. By following this rule you will never get lost: you will always return to the starting point eventually. However, as can be seen by our example, it does not guarantee that you will visit the entire maze.

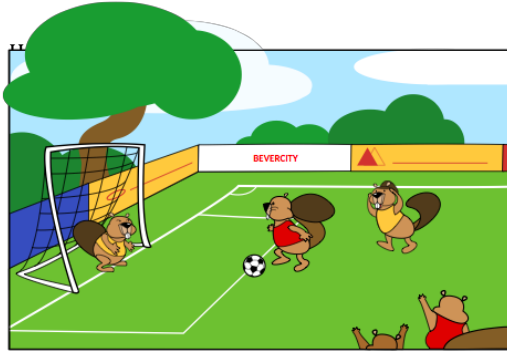
## Soccer Game

Castors:  
Benjamins: A  
Cadets:

Juniors:  
Seniors:



The Beaver Rangers have been playing a soccer game against the Forest Raiders.



### Question:

If we know that only one team manages to score two goals in a row, which of the following cannot be the final score?

3-3 5-1 2-4 or 4-2

### Answer:

5-1

### Explanation:

There are three possible final standings in this game. If one team scores first and the other team scores 2 in a row the final standing will be 3-3. If the first team scores first and also scores two in a row then the final standing will be 4-2. If the second team scores first and also scores two in a row then the final standing will be 2-4.

### It's Computational Thinking:

*CT Skills - Algorithmic Thinking (AL), Evaluation (EV)*

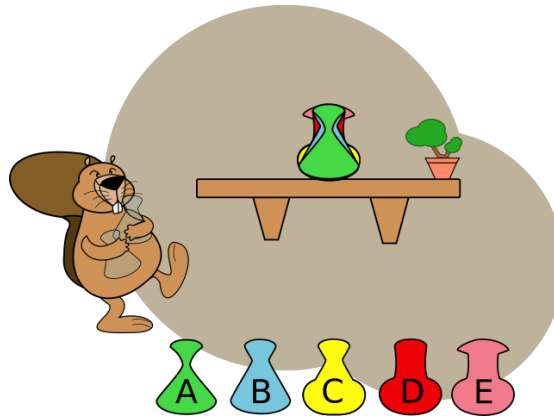
*CS Domain - Algorithms and programming*

*Tags - IF conditions*

The solution to this problem is a nested if statement: The first if statement checks which team scores first. The second if statement checks if a team scored two goals in a row.



A Beaver puts five bottles on a table.  
He places them so that every bottle has a bit showing.  
He places the first bottle at the back of the table and puts each new bottle in front of those already placed.



**Question:**

In what order are the bottles placed when they appear as shown in the picture?

- E D C B A
- D B C A E
- E C D A B
- D C E B A

**Answer:**

E D C B A

**Explanation:**

You can try to solve this different ways. If you figure out that the thin bottle should be at the front otherwise it will disappear behind one of the other bottles, you already know that A has to be in front. You can try that with each bottle in turn until you solve the task. You can also check which bottle is large at the top or middle, since in those places the bottles differ most. Small bottles need to be in front.

**It's Computational Thinking:**

*CT Skills - Abstraction (AB), Evaluation (EV)*

*CS Domain - Data, data structures and representations*

This is basically a sorting problem. You are asked to sort the bottles in a specific way. Here, shapes and sizes are important. One has to decide the ordering according to these properties.

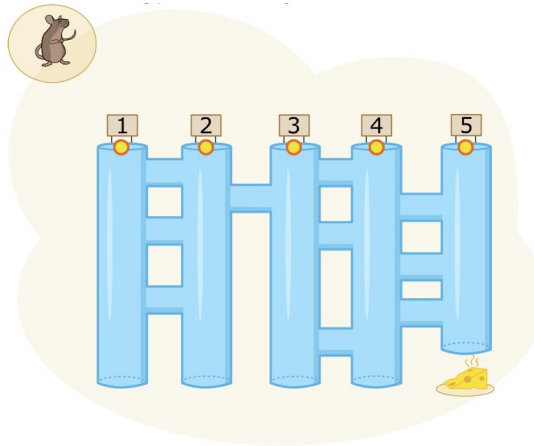
# Tube System

Castors: B Juniors:  
Benjamins: A Seniors:  
Cadets:



A mouse is at the entrance of a tube system. It wants to reach the cheese at the end of tube 5.  
The mouse always follows these commands:

1. Go downwards until a crossing
2. At the crossing, move through to the next vertical tube
3. Go to command 1



### Question:

In which tube should the mouse start so that it reaches the cheese?

1 2 3 4 or 5

### Answer:

3

### Explanation:

From tube 1 the mouse always reaches tube 3.

From tube 2 it reaches tube 1.

From tube 4 it reaches tube 2.

From tube 5 the mouse gets to tube 4.

### It's Computational Thinking:

*CT Skills - Algorithmic Thinking (AL)*

*CS Domain - Algorithms and programming*

Many robots are programmed so that they have to follow exact commands. This mouse does the same thing: it follows the commands 'go downwards' and 'change directions at the next crossing' over and over again. These kind of commands depend on the choice of the tube entrance as to which way the mouse runs in the tube system. Most computer programs are deterministic: if you always input the same data, the program performs the same calculations and delivers the same output.



## Party Guests

Castors: B Juniors:  
Benjamins: A Seniors:  
Cadets: A



To arrange a dinner party Sara the beaver needs to talk to five friends:

Alicia, Beat, Caroline, David and Emil.

Sara can talk to Emil right away. However, to talk to her other friends, there are a few points to consider:

- 1- Before she talks to David, she must first talk to Alicia.
- 2- Before she talks to Beat, she must first talk to Emil.
- 3- Before she talks to Caroline, she must first talk to Beat and David.
- 4- Before she talks to Alicia, she must first talk to Beat and Emil.

### Question:

In what order should Sara talk to all of her friends if she wants to talk to all of them?

Drag the names into the right order.

Alicia	Beat	Caroline	David	Emil
⇒	⇒	⇒	⇒	

# Secret Recipe

Castors: C Juniors:  
Benjamins: B Seniors:  
Cadets:

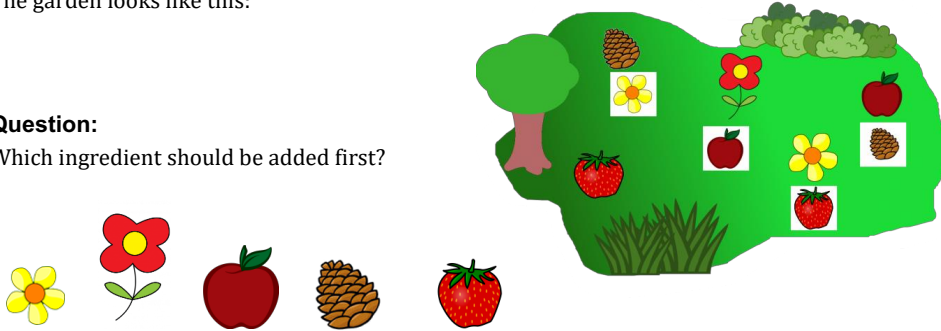


Eszter has asked István to cook a special cake made of five ingredients. She has put labels next to the ingredients in the garden. One ingredient has no label. The labels tell István in what order the ingredients must be added.

The garden looks like this:

### Question:

Which ingredient should be added first?



### Answer:

### Explanation:



If Eszter starts with the flower, she can add all five ingredients in the right order. The first added ingredient must be the one with no referring image.

Choosing the strawberry, she could not have continued to the next as there is no paper with it. The apple is not correct because if she had started with the apple, she would have skipped the red flower. The pine cone is not correct because if she had started with the cone, she would have skipped the red flower and the apple.

### It's Computational Thinking:

*CT Skills - Algorithmic Thinking (AL), Decomposition (DE)*

*CS Domain - Data, data structures and representations*

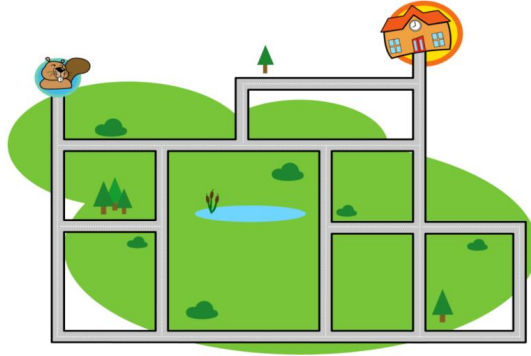
The data structure used here is called a linked list in which there can be an arbitrary number of items. A linked list is a linear collection of data elements that consist of an item and a reference point (pointer) showing the next item. The first item of the linked list is very important as the list starts from there and it is the only point that refers to the whole list.

The recipe here is a linked list. The ingredients are the items and each slip of paper is the pointer to the next item in line. In other words the plants are the data and the slips of paper are the pointers. The first component is that ingredient which is not referred by any paper, but accompanied by a paper.

The benefit of the linked list is that items of different types and sizes can be stored together, just like fruits and flowers in this question.

# Car Trip

Castors: C Juniors:  
Benjamins: B Seniors:  
Cadets: A



A self-driving car needs to take a student to school.

The car is programmed so that it only use these 3 instructions:

**Left:** turn 90° left

**Right:** turn 90° right

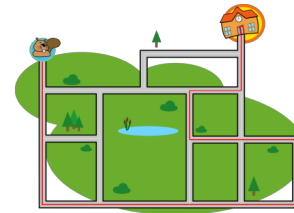
**Forward:** go forward until you cannot go forward anymore



## Question:

Write a set of instructions (a program) that will get the beaver to his school. You can do this by dragging the three instruction blocks next to the car.

## Answer:



## Explanation:

The important thing for participants to remember is that there is no forward movement when turning 90 degrees, so the 'straight' command has to be entered between every turn.

## It's Computational Thinking:

*CT Skills - Algorithmic Thinking (AL), Decomposition (DE)*

*CS Domain - Algorithms and programming*

The task is similar to giving instructions to a robot: writing a computer program requires step by step execution. Programs are essential to our use of computers: they tell computers what sequences of operations they must do. Computers and robots are good at computing fast, doing repetitive things, but they cannot think just by themselves, and require instructions to perform tasks. As shown in this task, the order of the operations is very important: the right set of instructions in the wrong order will not give the expected result.

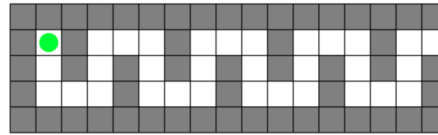
# Robot Exit

Castors: C Juniors  
Benjamins: B Seniors:  
Cadets: A



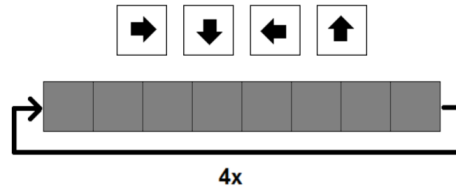
Help the green robot to exit the maze.

The robot will repeat your instructions 4 times.

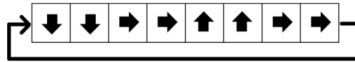


### Question:

Drag the arrows to form a set of instructions.



### Answer:



### Explanation:

In mobile robotics, maze problem solving is one of the most common problems. To solve this problem, an autonomous robot is used. Mazes can be of different kinds; having loops, without any loops, grid systems or without a grid system. In this short loop maze algorithm, the robot is instructed to follow a preference of directions.

### It's Computational Thinking:

*CT Skills - Algorithmic Thinking (AL)*

*CS Domain - Algorithms and programming*

In computer programming, a loop is a sequence of instruction's that is continually repeated until a certain condition is reached.

Typically, a certain process is done, such as getting an item of data and changing it, and then some condition is checked such as whether a counter has reached a prescribed number.

If it hasn't, the next instruction in the sequence is an instruction to return to the first instruction in the sequence and repeat the sequence. If the condition has been reached, the next instruction "falls through" to the next sequential instruction or branches outside the loop.

A loop is a fundamental programming idea that is commonly used in writing programs.

# Party Banner

Castors: C Juniors:  
Benjamins: B Seniors:  
Cadets: A



Beaver Bert has a long strip of colored paper for a party.  
The strip has three different colors (yellow, red, blue) in a regularly repeating pattern.  
Bert's friend, James, has cut out a section of the paper, as shown in the diagram below.



James says that he will give back the missing piece of paper if Bert can correctly guess the size of the piece cut out.

### Question:

How many colored squares does the missing piece of paper have?

31 32 33 or 34

### Answer:

31

### Explanation:

We know the pattern ended with YRR, meaning that the James has cut out at least one B. After that, he cuts out some number of sequences of 4 (i.e., YRRB). After that, the right side of his piece of paper must have YR, since the second piece begins with RB. So, the length of his piece of paper is 1 (for B) +  $4 * X$  (where X is the number of repeated patterns YRRB) + 2 (for the YR). So, the length of her paper is  $4X + 3$ .

Looking at the possible answers, we see that 31/4 has remainder 3: that is,  $31 = 4 * 7 + 3$ . So, our equation is solved when  $X = 7$ . None of the other answers can be written as  $4X + 3$ .

### It's Computational Thinking:

*CT Skills - Abstraction (AB), Evaluation (EV), Generalisation (GE)*

*CS Domain - Algorithms and programming*

Finding a pattern in information is important for a variety of problems. For instance, sequences of DNA are composed of patterns, and finding repetitions or substrings that satisfy a certain property is an important research area in genetics and medicine. To solve these sorts of problems, we use text processing algorithms and pattern-matching programs to help determine whether certain strings appear in a sequence of text.

This problem also involves some abstraction and generalization: We take a sequence of information and generalize it into a formula or equation which we can solve. In order for computer scientists to solve problems, they need to take an explanation and convert it into something more concrete, formalized and mathematical in order to write a program to solve it.

# Beaver Code

Castors: B Juniors:  
Benjamins: B Seniors:  
Cadets: A
















Barbara has been given two stamps.

With one she can produce a little flower, with the other a little sun.

Being a clever girl, she thinks of a way to write her own name by using the code below:



Letter	B	A	R	E	Y
Code			  	   	   

So Barbara” becomes:

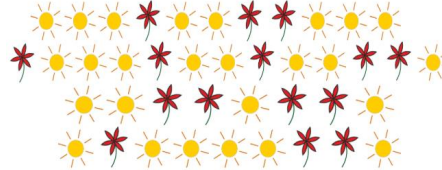


She then writes the names of her friends. Unfortunately they all got mixed up.

### Question:

Drag the sun-flower-codes to the names of her four friends.

Abby	
Arya	
Barry	
Ray	



# Beaver Code



**Answer:**

3

**Explanation:**

This problem is most easily solved by noting that Abby starts with an A and a B and so we look for a code with two suns and a flower at the start. There is only one of these so this is assigned. Next it is noted that Arya's code begins with three suns and a flower. Again there is only one of these so this is assigned. By continuing in this way, all the codes are quickly assigned to the correct names.

**It's Computational Thinking:**

*CT Skills - Algorithmic Thinking (AL), Decomposition (DE), Generalisation (GE)*

*CS Domain - Data, data structures and representations*

*Tags -*

Often in Computational Science, instead of storing data in a simple and straightforward way, we can devise a scheme to store it more efficiently, using less space.

For instance, computers store information about characters that can be typed on the keyboard in what is called an ASCII encoding. Each letter corresponds to a different sequence of 8 bits (0's and 1's). In ASCII, every character takes the same amount of space.

However, letters have different frequencies in texts (for example, the letter "E" is the most common letter in English words), and we can use these frequencies to improve our encoding.

Specifically, we encode frequent letters with smaller codes: in this question, B should be frequent and takes one symbol, A two, and the other letters more. There is a famous and widely used algorithm to do this for texts, named the Huffman coding. You cannot however use any encoding you wish: you have to make sure the code is unambiguous. For example, suppose the code was the following: Letter B was one flower, letter A was two flowers.

What do two flowers mean? It could be BB or A, but we have no way to know which one for sure. One way to achieve unambiguity is to make sure the code is prefix-free; that is if we take the code of a letter, it is not the beginning of any other code.

Also, since the Huffman code used depends on the text itself (it depends on the frequencies of letters), it is necessary to store the correspondence between the code and the actual letters. This takes a bit of space, but is negligible for long texts.

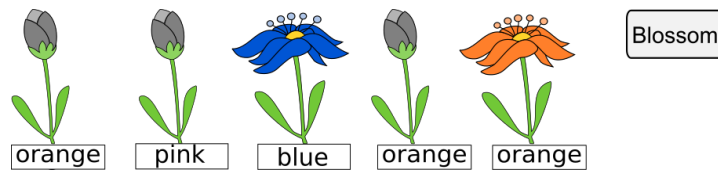


Jane is playing a computer game.

First the computer secretly chooses colors for five buds. The available colors for each flower are **blue**, **orange**, and **pink**. Jane has to guess which flower has which color. She makes her first five guesses and presses the Blossom button.

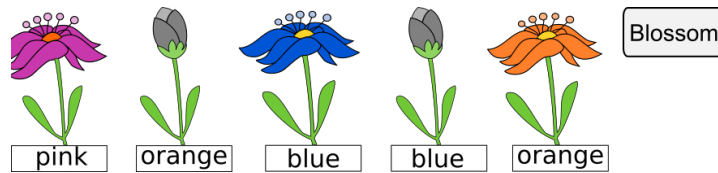
The buds, whose colors she guessed correctly, break into flowers. The others remain as buds.

Jane's first go:



Jane then has another go at guessing and presses the Blossom button again.

Jane's second go:



**Question:**

What colors did the computer choose for the flowers?

- A. blue pink blue orange orange
- B. pink blue blue blue orange
- C. pink blue blue pink orange
- D. pink pink blue pink orange





**Answer:**

C. pink blue blue pink orange

**Explanation:**

After two guesses there are three blossomed flowers. So we can already see the color chosen by the computer for the first, third and fifth flower. The color of the first flower is pink, so answer A) cannot be correct.

For the second flower Jane guessed pink in the first guess and it did not blossom, then she guessed orange and it did not blossom either. As there are only three colors available, the second flower must be blue. This rules out answer D).

Similarly, Jane chose orange and blue for the fourth flower and it still has not blossomed, so it must be pink. And this rules out answer B)

Answer C) must therefore be correct.

**It's Computational Thinking:**

*CT Skills - Evaluation (EV), Generalisation (GE)*

*CS Domain - Algorithms and programming*

Drawing consequences from events that happened or did not happen is an important ability for solving many kinds of problems. The task is a simplified version of the Mastermind board game. It is simplified because after each guess the player gets complete information about all the flowers. If in each guess the player chooses a different color for the not-yet-blossomed flowers, then in the third guess he/she can always correctly pick the colors of all the flowers.

## ☐ Magic Potions

Castors:  
Benjamins: C  
Cadets: B

Juniors: A  
Seniors: A



Betaro Beaver has discovered five new magic potions:

- one makes ears longer
- another makes teeth longer
- another makes whiskers curly
- another turns the nose white
- the last one turns eyes white.

Betaro put each magic potion into a separate beaker. He put pure water into another beaker, so there are six beakers in total. The beakers are labeled A to F. The problem is, he forgot to record which beaker contains which magic potion!

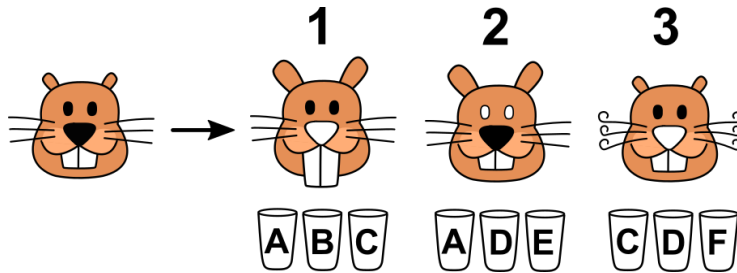


To find out which potion is in each beaker, Betaro set up the following experiments:

Expt 1: A beaver drinks from beakers A, B and C together - the effects are shown in Figure 1.

Expt 2: A beaver drinks from beakers A, D and E together - the effects are shown in Figure 2.

Expt 3: A beaver drinks from beakers C, D and F together - the effects are shown in Figure 3.



**Question:**

Which beaker contains pure water?



**Answer:**

Beaker D

**Explanation:**

Solution 1:

By Experiment 1, none of A, B and C is pure water, since there are three changes that happen to the beaver.

By Experiment 2, either D or E is pure water or the magic potion making his nose white since A is not pure water, from Experiment 1.

By Experiment 3, D and F are pure water or the magic potion making his whiskers curly, since C is not pure water, again from Experiment 1.

Therefore, D is pure water.

Solution 2:

Experiment 1 has three effects, Experiment 2 and 3 both have two effects. Therefore, there is no pure water in Experiment 1 and there is exactly one water beaker in Experiment 2 and Experiment 3. The only common beaker between experiments 2 and 3 is beaker D. Thus, D is pure water.

**It's Computational Thinking:**

*CT Skills - Algorithmic Thinking (AL), Evaluation (EV)*

*CS Domain - Algorithms and programming*

In this problem we have a collection of facts that we need to find new information from. This can be done using logical reasoning. Logic plays an important role in Computer Science. The smallest unit a computer works with is a bit, which has a value of 1 (true) or 0 (false). All other information in a computer is stored using a specific combination of bits. The computer uses logic to figure out what decisions it should follow, and each of these decisions is based on whether certain bits are set to true (1) or false (0).

This problem also explores basic set theory. We are looking for an element in the set which is not in the set used in Experiment 1, which means it is the complement of A, B, C. We then look at the intersection of Experiments 2 and 3 in order to determine the common element in both.

## ■ ■ Hurlers Shake Hands

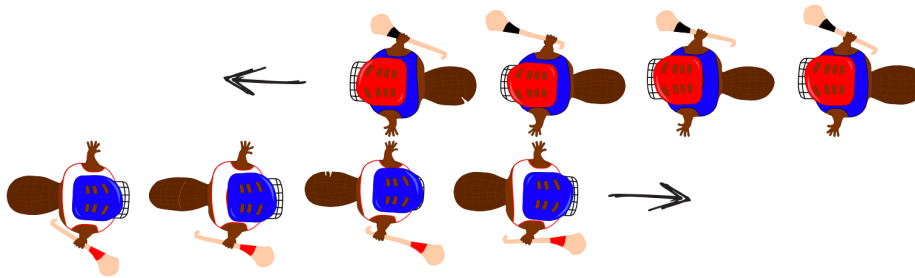
Castors: Juniors: A  
Benjamins: C Seniors:  
Cadets: B



Beavers enjoy playing hurling.

After the game ends, the beavers in each of the two teams line up in a row and walk past the other team. As they pass each other, they shake hands. At the beginning, only the first player on each team shakes hands. Next, the first two players shake hands (see picture below). This continues until each player has shaken hands with every player on the other team.

There are 15 players on each team.



### Question:

If each player takes one second to shake hands and move to the next player, how many seconds of shaking hands will there be?



**Answer:**

29

**Explanation:**

The amount of handshaking is exactly the length of one line plus the length of the other line, minus one.

Let us imagine that there is only 1 player on each team. After 1 second, all handshaking has finished. Let us imagine that there are only 2 players on each team. During the first second, the first player on each team shakes hands. During the second second, the first player on each team is shaking hands with the second player on the other team, and during the third second, the second two players are shaking hands with each other. So, that's three seconds.

With 15 players in each team, the number of seconds required is  $15 + 15 - 1 = 29$ .

**It's Computational Thinking:**

*CT Skills - Algorithmic Thinking (AL)*

*CS Domain - Computer processes and hardware*

*Tags - Parallel processing*

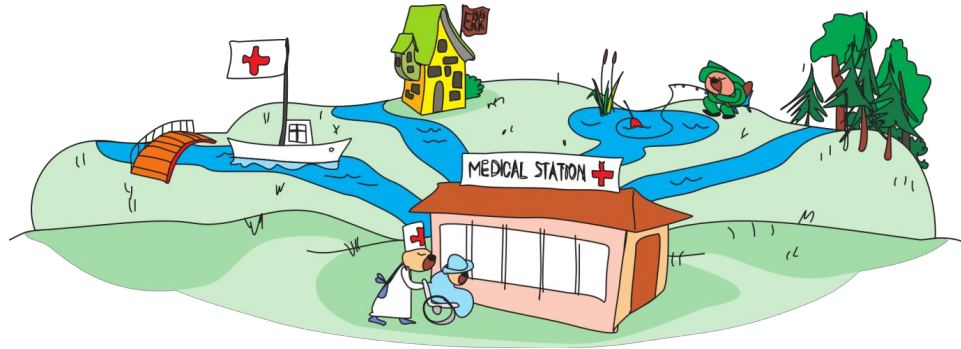
This task can be viewed as an illustration of a parallel processing paradigm called pipeline processing. Pipeline processing is a very efficient way to get many computers working together to solve problems quickly, but it can take a relatively long time to reach that efficiency, just like our players at the back of each line had to wait quite a while before their first handshake.

Analysing the running time of an algorithm is a sophisticated part of computer science called computational complexity analysis. In this Task, we know the team size is fixed at 15, and can deduce that the "running time" of the hand shaking algorithm is 29 seconds. However, in computational complexity we would be asked to measure the running time independent of a specific team size. We would conclude that the hand shaking algorithm takes  $2N-1$  seconds, for any team size  $N$ , where  $N$  is 1, or any larger natural number.

# Primary Health Care

Castors:  
Benjamins: C  
Cadets: B

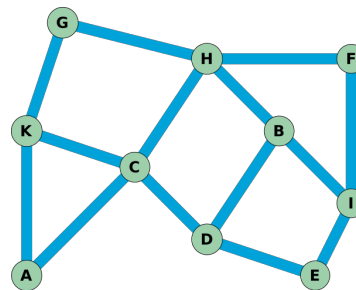
Juniors: A  
Seniors:



Doctor Hamid wants to build three hospitals for the beavers.  
The hospitals can only be built on the places shown on the map below.  
To get to a hospital, the beavers should not have to swim through more than one stream from any of these places.

**Question:**

Choose three places to build the hospitals for Doctor Hamid.





**Answer:**

There are several correct solutions, one for instance uses the places E, H and K:

- For the places D, E and I the beavers can swim to E.
- For the places B, C, F, G and H the beavers can swim to H.
- For the places A, C, G and K the beavers can swim to K.

The other solutions are: A E H, C G I, C H I, C I K, C E H and D F K.

**Explanation:**

The solutions can be found by placing a station at a random position and marking all stations that are reachable within one step. Then you can position the next station and so on. Once all three stations are placed there are two possibilities: either it's a solution or there are one or more places that are not marked. If it's not a solution, you can remove the last station you've placed and place it in another place and check again. If you are still not lucky to find a solution with 3 stations you have to "backtrack" and place the last station on another place. By doing this systematically one can find all possible solutions.

**It's Computational Thinking:**

*CT Skills - Abstraction (AB), Evaluation (EV)*

*CS Domain - Data, data structures and representations*

In Computing this channel system is generalized to the concept of a graph consisting of vertices (intersections) and edges (water canals). The more general problem is to find a so-called "vertex cover" in a graph. This is a subset of the vertices that cover the whole graph. Whenever all the neighbor vertices to this subset are added together, they will cover all vertices of the graph. Usually a minimal number of such vertices is the most cost efficient. In more complex graphs it is very hard to find these cost effective subsets of the vertices. It needs a computer algorithm to find a solution.

The method of placing the stations described in the explanation section is called backtracking. You try one solution and if it is not correct you take back the last step you've made and try another step, ideally systematically until you have exhausted all possible last steps. Then you take back the pre-last step, try all solutions and so on until you have found a correct solution. This method is not very efficient, but for this kind of problems it works reliably.

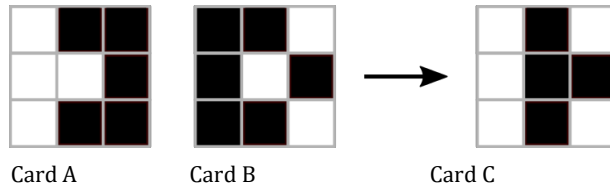
# Paint it Black

Castors:  
Benjamins: C  
Cadets:

Juniors:  
Seniors:

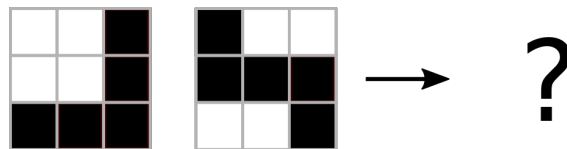


Combining Card A and Card B, you get Card C:



**Question:**

How many black cells will Card F have after combining Card D and Card E?



**Answer:**

3

**Explanation:**

Combining the cards obeys the following rule. When the color of the corresponding cells is the same the resulting color is black. Otherwise the resulting color is white.



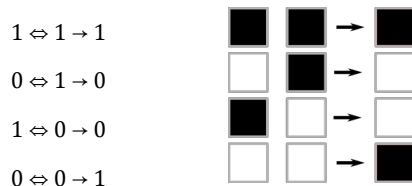
**It's Computational Thinking:**

CT Skills - Abstraction (AB), Algorithmic Thinking (AL), Evaluation (EV)

CS Domain - Algorithms and programming

Tags - Boolean algebra

A Boolean circuit is an example of a mathematical computation model. An equivalence is one of the basic Boolean operations. If the white cell is interpreted as 0 or FALSE and the black cell as 1 or TRUE, this operation could be described this way:





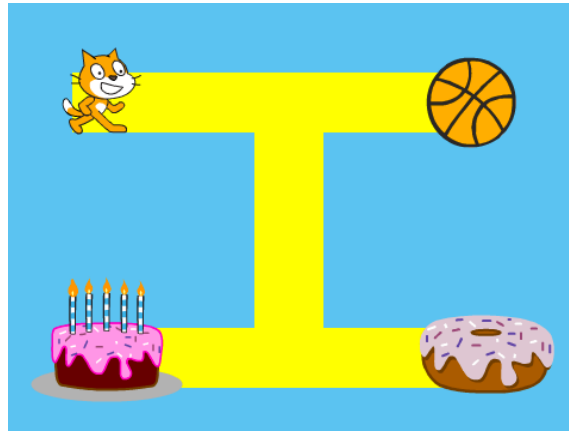
## APPENDIX E

### PROGRAMMING CONCEPTUAL KNOWLEDGE POST-TEST

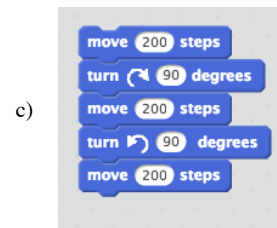
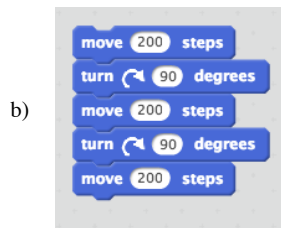
Name: \_\_\_\_\_

Please complete the following items. If you have questions please raise your hand.

#### Question 1

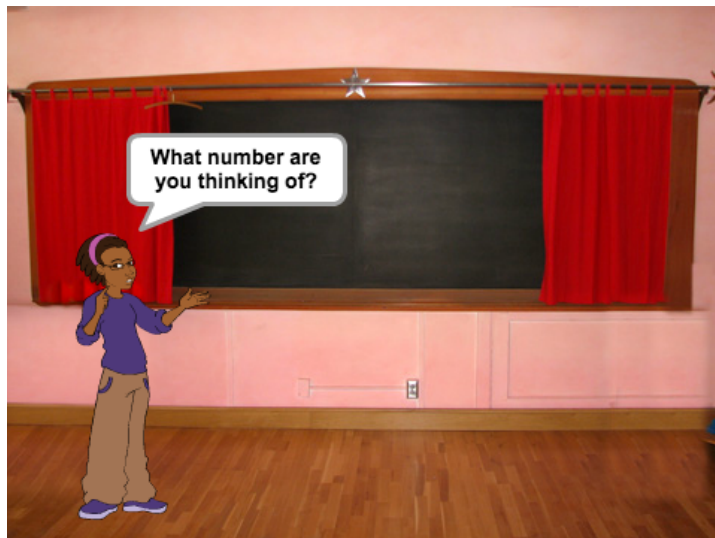


Which of the Scratch programs below will get the Scratch Cat to the donut? (circle your answer)



f) I don't know.

## Question 2



Look at the script below:

```
when clicked
ask "What number are you thinking of?" and wait
if answer > 20 then
say "Great!" for 2 secs
else
say "Awesome!" for 2 secs
```

Which number could you enter to make the teacher say "Great!""? (Circle your answer)

a) -5

b) 20

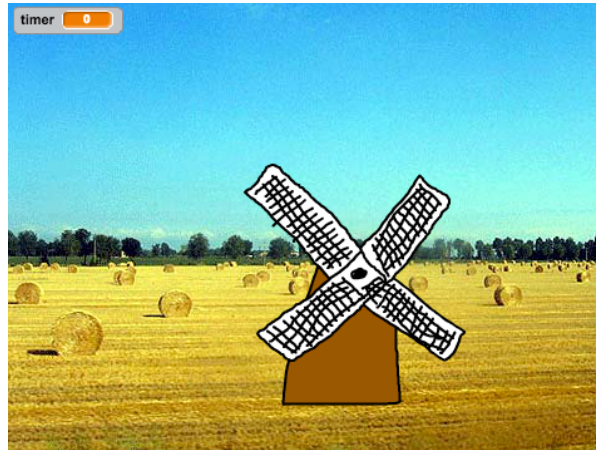
c) 19

d) 21

e) 0

f) I don't know

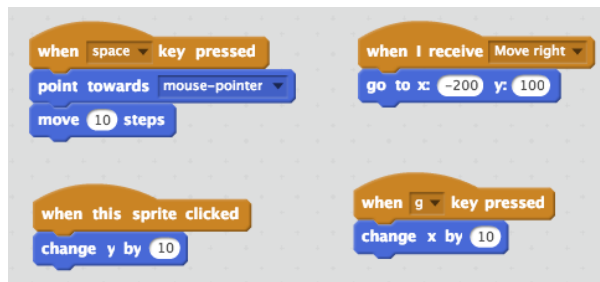
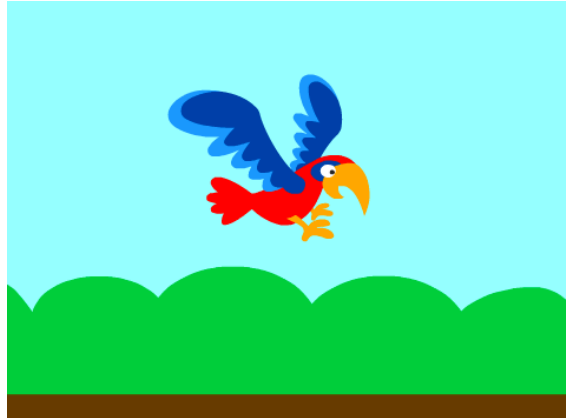
Question 3



When the green flag is clicked the blades on this windmill turn clockwise. Which of the below sequences would make the windmill rotate the **longest**? (Circle your answer)

- a)
- b)
- c)
- d)
- e)
- f) I don't know.

Question 4



Which of the following would you need to do to move the parrot to the right? (Circle your answer)

- a) Do nothing
- b) Click the parrot
- c) Press the space bar
- d) Make some noise
- e) Press the "g" key
- f) I don't know.

**Question 5**

In the program below, the dinosaur should say the 8 times table, however there's a problem with the code and the dinosaur repeatedly says "8".



```
set number to 1
repeat until number > 12
  say number * 8 for 1 secs
```

Which script contains the correct code to make the dinosaur say the complete 8 times table? (Circle your answer)

a)

```
set number to 1
repeat until number > 12
  say number * 8 for 1 secs
  wait 1 secs
```

b)

```
set number to 1
repeat until number > 12
  say number * 8 for 1 secs
  change number by 1
```

c)

```
set number to 1
repeat until number > 12
  say number + 1 * 8 for 1 secs
```

d)

```
set number to 1
repeat until number < 12
  say number * 8 for 1 secs
```

e)

```
set number to 1
repeat until number > 12
  change number by 1
  say number * 8 for 1 secs
```

f) I don't know.

**Question 6**

In this Scratch program, the polar bear says to Khalid, "Let me see you dance!" using the **Bear Script** code below. Khalid's dance is controlled by the **Dance Script** code below.

**Bear Script**

```

when green flag clicked
wait 1 secs
say "Let me see you dance!"
wait 1 secs
broadcast dance
    
```



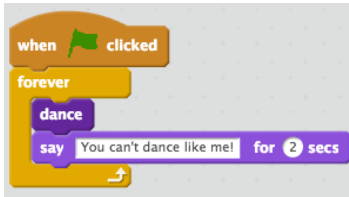
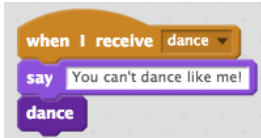
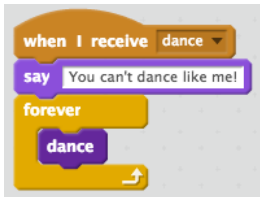
**Khalid's Dance Script**

```

define dance
switch costume to khalid-a
wait 0.2 secs
switch costume to khalid-b
wait 0.2 secs
switch costume to khalid-c
wait 0.2 secs
switch costume to khalid-d
wait 0.2 secs
    
```

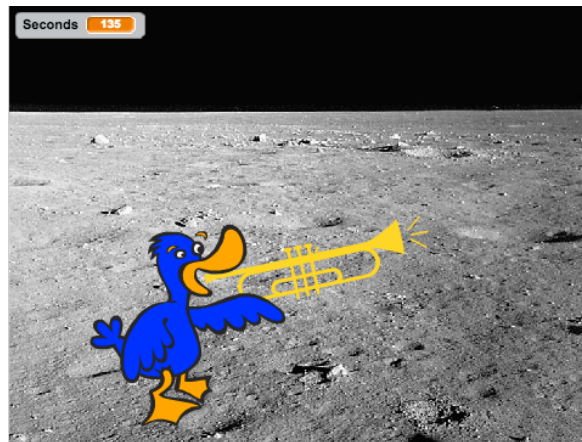


Which script contains the correct code to make Khalid both **dance** and **say**, "You can't dance like me!" forever?  
(Circle your answer)

- a) 
- b) 
- c) 
- d) 
- e) 
- f) I don't know.

**Question 7**

This moon duck is going to play a Moon Song on the trumpet. His song is exactly **2 minutes and 15 seconds (135 seconds)** long. The "Seconds" timer in the upper right corner of the backdrop should count the seconds so that other moon ducks can appear on the screen at a specific time in the song, however, something isn't quite working right. As soon as the green flag is clicked, the seconds timer goes from 0 to 135 instead of counting up by 1 every second.



Which is the correct sequence of code to play the Moon Song and make the Seconds timer change by one every second in the song? (Circle your answer)

- a)
- b)
- c)
- d)
- e)
- f) I don't know.

APPENDIX F

CORRELATIONS BY OVERALL SAMPLE

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1. Sex	—																
2. Age	.290	—															
3. Grade	.190	.896**	—														
4. SpEd Status	.187	-.032	-.195	—													
5. PSSA ELA	.405*	-.196	-.025	-.485*	—												
6. PSSA Math	.030	-.207	-.034	-.477*	.533**	—											
7. Attendance	-.310	-.200	-.222	.182	.181	-.327	—										
8. Instructional Hrs	-.315	-.288	-.253	.198	.192	-.243	.976**	—									
9. Chips	.054	.111	-.083	.013	-.144	-.524*	.745**	.542*	—								
10. PCK <sub>1</sub>	.322	.032	.039	-.378	.227	.323	.302	.341	.029	—							
11. PCK <sub>2</sub>	.235	.220	.275	-.177	.308	-.057	.190	.171	.135	.239	—						
12. WJ-IV CF <sub>1</sub>	.485*	.286	.301	-.042	.062	.228	-.257	-.185	-.181	.502*	.229	—					
13. WJ-IV CF <sub>2</sub>	.476*	.322	.312	-.247	.137	.017	-.193	-.143	-.103	.497	.411	.751**	—				
14. KTEA-3 MCA <sub>A</sub>	.299	.204	.338	-.439	.462*	.786**	-.301	-.192	-.495	.620**	-.147	.569*	.288	—			
15. KTEA-3 MCA <sub>B</sub>	.305	.004	.085	-.694**	.607**	.831**	-.212	-.171	-.079	.777**	.231	.446	.316	.928**	—		
16. Type T	-.191	-.082	.083	-.010	.023	.281	-.164	-.011	-.513*	-.054	-.035	.317	.065	.325	-.144	—	
17. CPS <sub>1</sub>	.164	.157	.252	-.330	.488*	.526*	-.209	-.143	-.399	.230	.177	.510*	.119	.799**	.788**	.278	—
18. CPS <sub>2</sub>	.077	.597*	.680**	-.522*	.544*	.518*	-.276	-.249	-.292	.720**	.284	.454	.195	.655*	.681**	.043	.712**

Note: \* p < .05; \*\* p < .01



APPENDIX G

CORRELATIONS BY CONTROL GROUP

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1. Sex	—																
2. Age	.359	—															
3. Grade	.190	.929**	—														
4. SpEd Status	-.158	.000	.103	—													
5. PSSA ELA	-.462	-.509	-.352	-.389	—												
6. PSSA Math	-.102	-.371	-.244	-.518	.637*	—											
7. Attendance	-.471	-.417	-.274	.165	.426	.096	—										
8. Instructional Hrs	-.471	-.417	-.274	.165	.426	.096	1.00	—									
9. Chips	.338	-.161	-.168	-.715*	.258	.540	.640	.640	—								
10. PCK <sub>1</sub>	.109	-.214	-.421	-.756*	.333	.214	.349	.349	.800*	—							
11. PCK <sub>2</sub>	-.104	.092	-.033	-.696	.880*	.516	.216	.216	.226	.667	—						
12. WJ-IV CF <sub>1</sub>	.457	.216	.219	-.058	-.287	-.304	.065	.065	.673	.571	-.224	—					
13. WJ-IV CF <sub>2</sub>	.516	.433	.359	-.588	.061	-.279	.025	.025	.801	.736	.277	.677	—				
14. KTEA-3 MCA <sub>A</sub>	.433	-.126	-.020	.000	.107	.643	.541	.541	.698	.086	-.224	.306	-.559	—			
15. KTEA-3 MCA <sub>B</sub>	.262	-.168	-.281	-.756*	.500	.905**	.048	.048	.741	.543	.577	.029	.109	.900*	—		
16. Type T	.088	-.304	-.280	.681*	-.093	-.228	.239	.239	-.296	-.279	-.216	.085	-.258	.147	-.559	—	
17. CPS <sub>1</sub>	.453	-.048	.031	.000	.012	.060	.207	.207	.616	.000	-.258	.446	-.205	.886*	.667	-.147	—
18. CPS <sub>2</sub>	-.293	.493	.772	-.393	.257	.543	.348	.348	.395	.100	.395	.300	.116	.100	.314	-.527	-.400

Note: \* p < .05; \*\* p < .01

APPENDIX H

CORRELATIONS BY EXPERIMENTAL GROUP

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1. Sex	—																
2. Age	.418	—															
3. Grade	.233	.935**	—														
4. SpEd Status	.529	-.228	-.480	—													
5. PSSA ELA	-.123	.004	.141	-.649*	—												
6. PSSA Math	.025	.030	.115	-.519	.849**	—											
7. Attendance	-.105	-.139	-.202	.173	-.347	-.455	—										
8. Instructional Hrs	-.105	-.139	-.202	.173	-.347	-.455	1.00	—									
9. Chips	.284	.231	.094	.522	-.699*	-.851**	.546	.546	—								
10. PCK <sub>1</sub>	.390	.417	.298	-.075	.263	.201	.513	.513	-.079	—							
11. PCK <sub>2</sub>	.519	.537	.519	.163	-.275	-.185	-.105	-.105	.330	-.140	—						
12. WJ-IV CF <sub>1</sub>	.520	.376	.376	-.098	.540	.558	-.508	-.508	-.178	.374	.552	—					
13. WJ-IV CF <sub>2</sub>	.360	.519	.359	.088	.098	-.043	-.408	-.408	.084	.093	.404	.667*	—				
14. KTEA-3 MCA <sub>A</sub>	.220	.389	.441	-.453	.753**	.736**	-.378	-.378	-.382	.562	-.294	.830**	.325	—			
15. KTEA-3 MCA <sub>B</sub>	.217	.395	.394	-.572	.828**	.853**	-.408	-.408	-.563	.867**	-.315	.723*	.132	.971**	—		
16. Type T	-.575	.183	.276	.596*	.429	.437	-.252	-.252	-.536	-.140	-.183	.029	-.031	.243	.287	—	
17. CPS <sub>1</sub>	-.120	.192	.363	-.671*	.943**	.752**	-.516	-.516	-.564	.142	-.073	.616*	.202	.800**	.838**	.450	—
18. CPS <sub>2</sub>	.034	.618	.624	-.572	.762*	.592	-.472	-.472	-.467	.758*	.222	.715*	.204	.812**	.731*	.244	.922**

Note: \* p < .05; \*\* p < .01