SECONDARY SCHOOL STUDENTS' PROGRAMMING AND COMPUTATIONAL THINKING SKILLS: TRADITIONAL AND INTERDISCIPLINARY APPROACHES TO TEACHING PROGRAMMING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

RÜKİYE ALTIN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER EDUCATION AND INSTRUCTIONAL TECHNOLOGY

FEBRUARY 2021

Approval of the thesis:

SECONDARY SCHOOL STUDENTS' PROGRAMMING AND COMPUTATIONAL THINKING SKILLS: TRADITIONAL AND INTERDISCIPLINARY APPROACHES TO TEACHING PROGRAMMING

submitted by **RÜKİYE ALTIN** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Education and Instructional Technology, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Dr. Hasan Karaaslan
Head of Department, **Computer Edu. and Inst. Tech.** _____

Assoc. Prof. Dr. Tuğba Tokel
Supervisor, **Computer Edu. and Inst. Tech., METU** _____

Examining Committee Members:

Prof. Dr. Yasemin Gülbahar Güven
Computer Edu. and Inst. Tech, Ankara University _____

Assoc. Prof. Dr. Tuğba Tokel
Comp. Edu. and Ins. Tech., METU _____

Prof. Dr. Ömer Delialioğlu
Comp. Edu. and Ins. Tech., METU _____

Asst. Prof. Dr. Cengiz S. Aşkun
Comp. Edu. and Ins. Tech., METU _____

Asst. Prof. Dr. Halil Ersoy
Comp. Edu. and Ins. Tech., Başkent University _____

Date: 15.02.2021

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name Last name : Rükiye Altın

Signature :

ABSTRACT

SECONDARY SCHOOL STUDENTS' PROGRAMMING AND COMPUTATIONAL THINKING SKILLS: TRADITIONAL AND INTERDISCIPLINARY APPROACHES TO TEACHING PROGRAMMING

Altın, Rükiye
Ph.D., Department of Computer Education and Instructional Technology
Supervisor: Assoc. Prof. Dr. Tuğba Tokel

February 2021, 264 pages

The focus on K-12 Computer Science education has increased in line with today's need for a population skilled in computational thinking, and with the growth in careers that are within or at least related to the field of computer science. As such, young learners should be introduced to computer science from an early age, as programming is now considered by many to be an indispensable change to traditional learning methods, and is seen as playing a significant role in the future career choices of most school-aged learners. The current study aims to serve two main purposes. First, the study focused on the effects of teaching programming, through both a traditional and an interdisciplinary approach, on secondary school students' programming skills and also on their computational thinking skills. Second, the study examined how to involve mathematics as a second discipline in the teaching of programming in order to assess the differences between programming skills acquisition and retention.

The study presents a Quasi-Experimental research conducted on a large sample (N = 188), with four randomly assigned experimental groups and four randomly assigned control groups. In this context, the study's data were collected using three different

scales; the Computer Programming Self-Efficacy Scale, the Computational Thinking Skills Self-Efficacy Scale, and the Mathematics Attitude Scale, in addition to an achievement test applied to students according to the programming language that they were taught. Interviews were conducted with both the students and their teachers in order to collect the study's qualitative data. The study was applied with two different sets of groups, with those in the experimental groups taught using an interdisciplinary approach, whilst those in the control groups were taught according to the traditional method. Both sets of groups were taught based on different lesson plans, with the experimental groups taught programming with the integration of mathematics, while the control groups were taught according to the traditional single-discipline method. In total, 12 ICT teachers and two mathematics teachers took an active part in the preparation of both lesson plans and the activities.

The study's results showed that teaching programming with mathematics integrated as an interdisciplinary approach increased both the students' programming and computational thinking skills. Additionally, the results showed that teaching programming with the integration of mathematics affected the students' learning processes. According to retention results having notably increased for those students who studied in the experimental groups, the learners maintained their newly learned programming knowledge within their long-term memory. The current study is therefore considered important in that it provides lesson plans for a secondary school programming course that has been shown to positively affect students' programming learning.

Keywords: Programming for Children, Computational Thinking Skills, Algorithm, Programming, Self-Efficacy, Visual Programming, Interdisciplinary Collaboration, Computer Science in K-12

ÖZ

ORTAOKUL ÖĞRENCİLERİNİN PROGRAMLAMA VE BİLGİ İŞLEMSEL DÜŞÜNME BECERİLERİ: PROGRAMLAMA ÖĞRETİMİNDE GELENEKSEL VE DİSİPLİNLER ARASI YAKLAŞIM

Altın, Rükiye
Doktora, Bilgisayar ve Öğretim Teknolojileri Eğitimi
Tez Yöneticisi: Doç. Dr. Tuğba Tokel

Şubat 2021, 264 sayfa

K-12'de Bilgisayar Bilimi eğitimine odaklanma, günümüzün bilgi işlemsel düşünme konusunda yetenekli bir nüfusa olan ihtiyacı ve bilgisayar bilimi alanı dâhilinde veya en azından alanla ilgili olan kariyerlerin büyümesiyle aynı doğrultuda artmıştır. Bu nedenle, programlamanın artık birçok kişi tarafından geleneksel öğrenme yöntemlerinde vazgeçilmez bir değişim olarak kabul edildiği ve okul çağındaki öğrencilerin çoğunun gelecekteki kariyer seçimlerinde önemli bir rol oynadığı anlaşıldığı için, genç öğrenciler, erken yaşlardan itibaren bilgisayar bilimi ile tanışmalıdır. Mevcut çalışma iki ana amaca hizmet etmeyi amaçlamaktadır. İlk olarak, çalışma, hem geleneksel hem de disiplinler arası bir yaklaşımla programlama öğretmenin ortaokul öğrencilerinin programlama becerileri ve ayrıca bilgi işlemsel düşünme becerileri üzerindeki etkilerine odaklanmıştır. İkinci olarak, çalışma, programlama becerilerini edinme ve kalıcı öğrenme arasındaki farkları değerlendirmek için matematiğin programlama öğretimine ikinci bir disiplin olarak nasıl dâhil edileceğini incelemiştir.

Çalışma, dört rastgele atanmış deney grubu ve dört rastgele atanmış kontrol grubu ile büyük bir örneklem (N = 188) üzerinde gerçekleştirilen Yarı Deneysel bir

araştırmayı sunmaktadır. Bu bağlamda, çalışmanın verileri, öğrencilere öğretilen programlama diline göre uygulanan bir kazanım testine ek olarak, Bilgisayar Programlama Öz-Yeterlik Ölçeği, Bilgi İşlemsel Düşünme Becerileri Öz-Yeterlik Ölçeği ve Matematik Tutum Ölçeği olmak üzere üç farklı ölçek kullanılarak toplanmıştır. Araştırmanın nitel verilerini toplamak için hem öğrencilerle hem de öğretmenleriyle görüşmeler yapılmıştır. Çalışma, deney gruplarında yer alanlara disiplinler arası bir yaklaşımın kullanılarak öğretildiği ve kontrol grubundakilere ise geleneksel yönteme göre öğretildiği iki farklı grup kümesi ile uygulanmıştır. Her iki grup kümesine de farklı ders planlarına göre öğretilmiş, deney gruplarına matematik entegrasyonu ile programlama öğretilirken, kontrol gruplarına geleneksel tek disiplin yöntemine göre öğretilmiştir. Toplamda 12 Bilişim Teknolojileri öğretmeni ve iki matematik öğretmeni hem ders planlarının hem de etkinliklerin hazırlanmasında aktif rol almıştır.

Yarı-deneysel tasarım metodu kullanılarak (N = 188) uygulanan bu çalışmada rastgele atanan dört deneysel grup ve dört kontrol grubu ile çalışılmıştır. Çalışmada elde edilen veriler; “Bilgisayar Programlama Öz Yeterlilik Ölçeği” , “Bilgi İşlemsel Düşünme Becerileri Öz Yeterlilik Ölçeği” ve “Matematik Tutum Ölçeği” olmak üzere üç farklı ölçek ve bir “Programlama Kazanım Testi” ile elde edilmiştir. Nitel verileri toplamak için hem öğrencilerle hem de öğretmenlerle görüşmeler yapılmış; çalışma, disiplinler arası iş birliği yaklaşımıyla programlamanın öğretildiği deneysel grup ve geleneksel öğretim yöntemleriyle programlamanın öğretildiği kontrol grubu olmak üzere iki grup ile yürütülmüştür.

Çalışmanın sonuçları, disiplinler arası bir yaklaşım olarak entegre edilmiş matematik ile programlama öğretiminin hem öğrencilerin programlama hem de bilgi işlemsel düşünme becerilerini artırdığını göstermiştir. Ayrıca, sonuçlar matematik entegrasyonu ile programlama öğretiminin öğrencilerin öğrenme süreçlerini etkilediğini göstermiştir. Deney gruplarında öğrenim gören öğrenciler için önemli ölçüde artan akılda tutma sonuçlarına göre, öğrenenler yeni öğrendikleri programlama bilgilerini uzun süreli belleklerinde muhafaza etmişlerdir. Bu nedenle mevcut çalışma, öğrencilerin programlama öğrenmelerini olumlu yönde etkilediği

gösterilen bir ortaokul programlama dersi için ders planları sağladığından dolayı önemli kabul edilmektedir.

Anahtar Kelimeler: Çocuklar için Programlama, Bilgi İşlemsel Düşünme, Algoritma, Programlama, Öz Yeterlilik, Görsel Programlama, Disiplinler Arası İş Birliği, K-12'de Bilgisayar Bilimleri

This thesis is dedicated to my family and to my nephew, Ali

ACKNOWLEDGMENTS

The writing of this thesis has been a long journey, and one for which I owe a debt of gratitude to many for their support. First, I would like to express my sincerest gratitude to my supervisor, Assoc. Prof. Dr. Tuğba Tokel, for her guidance and constant feedback throughout the process. I also would also like to acknowledge my co-advisors, Prof. Dr. Ömer Delialioğlu and Prof. Dr. Yasemin Gülbahar Güven, for guiding me with patience and encouragement whenever I have been in need. I would like to express my deep gratitude to the examination committee members, Dr. Halil Ersoy and Dr. Cengiz Aşkun, as well as to Prof. Dr. Soner Yıldırım for his invaluable advice, support, and mentoring. I would also like to thank the Computer Education and Instructional Technology department for providing an additional work environment for my studies.

I would like to express my special gratitude to the management of educational programs for permitting me to conduct this study, and to my colleagues Mehtap Gömeç, Seda Adıgüzel and Azad Işık for their support and effort along this journey.

I am so lucky to have friends who I refer to as my second family in this world. I would like to thank my lovely A.R.M. team members, Aydan Anıl Demirtaş, and Merve Aksu, for always supporting me, for believing in my success, and providing that warm hug whenever it was needed. Having a great teammate is also the key to success, and I am lucky to have a great friend, teammate, and soul-sister, and clearly the best thing that happened to me in my teaching career, Eylem Erkan İşler, and her family for always being with me, opening up their home whenever I need a talk, and for raising my spirits whenever I was down. I am grateful too to have a friend like Ümmühan Yılmaz as my colleague, and I would like to thank her for her warm smile and support in my studies, as well as for her mathematical experience and depth of knowledge. I would also like to thank Duygu and Mustafa Bektik for her advice as friends on how to take slow, deep breaths in order to cope with everything that such an academic venture entails. I would like to thank my dear friend Ayşe Günay

Gökben, Pınar Kaya and Yasaman Alioon who I am really lucky to meet during my ODTÜ life. I am happy to have friends who shared the same feeling with me through this journey.

On a personal note, I would like to thank Reyyan Ayfer and Can Uğur Ayfer, from Bilkent University. Reyyan Hocam has always acted as my role model and has continued to encourage and support me whenever I have been in need. She is the reason that I started my PhD, and is the reason for my being able to finally finish it too. Thanks to her, I have come to know and appreciate a great ACM family, so I would also like to thank the ACM-WE members for their support and understanding.

I believe I am truly blessed to have such a great family, and I am so grateful for their support, encouragement, tolerance, and love. Having them in my life is incomparable with anything else. Special thanks to my dearest father, İsmet Altın, and my mother, Zeliha Altın, who have stood by me always and for every sacrifice they have made in order for me to reach my goals by supporting me every step of my life. I am fortunate to have a great sister, Leyla Altın Atakan, and brother, Mehmet Melik Altın, who have always been there whenever I needed them. Finally, I would like to thank lovely Ali, my dear little nephew who has recently joined our family; he gives me the greatest feeling, makes me smile all the time, and just makes me feel the luckiest person alive. Thank you, my amazing family, friends, and colleagues, for your support, understanding and patience. I am happy to come to the end of this particular journey, and now cannot wait to see what life has in store for me next.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vii

ACKNOWLEDGMENTS xi

TABLE OF CONTENTS xiii

LIST OF TABLES xvii

LIST OF FIGURES xix

LIST OF ABBREVIATIONS xx

CHAPTERS 1

1 INTRODUCTION 1

1.1 Problem Statement 3

1.2 Significance of the Study 5

1.3 Research Question 6

1.4 Organization of the Study 7

2 LITERATURE REVIEW 9

2.1 What is Computational Thinking Skill? 9

2.1.1 Characteristics of Computational Thinking 10

2.2 Computer Programming Education 12

2.2.1 Programming Education for Early Ages 16

2.2.2 The Relation between Computational Thinking and Computer Science .. 20

2.2.3 Programming Tools for K-12 Students 23

2.3 Interdisciplinary Approach in Education 27

2.3.1 Interdisciplinary Collaboration of Mathematics and Computer Science .. 29

2.3.2 Integrating Computational Thinking in Mathematics Education 33

2.4 Summary of Literature 35

3 METHODOLOGY 39
3.1 Research Questions 39
3.2 Participants 40
3.3 Design of the Study 42
3.3.1 Experimental Research Application 45
3.4 Experimental Processes 45
3.4.1 Pilot Study 46
3.4.2 Main Study 48
3.5 Data Collection 50
3.5.1 Instruments 50
3.6 Data Analysis 56
3.6.1 Trustworthiness 56
3.6.2 Quantitative Data Analysis 57
3.6.3 Qualitative Data Analysis 59
3.7 Researcher Role and Bias 64
3.8 Summary of Methodology 65
4 FINDINGS 67
4.1 Quantitative Analysis Findings 67
4.1.1 Research Question A: Mathematics Attitude 68
4.1.2 Research Question B: Computational Thinking Skills Self-Efficacy 78
4.1.3 Research Question C: Computer Programming Self-Efficacy 85
4.1.4 Research Question D: Small Basic Programming Achievement 90
4.2 Qualitative Analysis Findings 96
4.2.1 Descriptive Analysis of Qualitative Data 96
4.2.2 Research Question E: Teachers' Thoughts on Teaching Visual Programming Tool 100

4.2.3 Research Question F: Students Thoughts about Visual Programming Tool 105

4.3 Summary of Findings 117

5 DISCUSSION AND CONCLUSION 123

5.1 Discussion 123

5.1.1 Teaching programming approaches and Mathematics Attitude 123

5.1.2 Computational Thinking Skills Self-Efficacy 126

5.1.3 Computer Programming Self-Efficacy 129

5.1.4 Small Basic Programming Achievement 132

5.1.5 Teachers' Thoughts on Teaching Small Basic 135

5.1.6 Students Thoughts about Small Basic Programming 138

5.2 Conclusion 143

5.3 Assumptions 145

5.4 Implications for teachers and instructional designers 146

5.5 Recommendation for Further Research 147

5.6 Limitations of the Study 147

REFERENCES 149

APPENDICES 173

A. Activity Plan 173

B. Lesson Plans 175

C. Visuals for Lessons 206

D. Guidance for Teachers 218

E. Instruments 219

F. Teachers' Effect 247

G. Permissions 255

H. Post-Research Information Form 261

CURRICULUM VITAE 263

LIST OF TABLES

TABLES

Table 1. Programming Tools for K-12 Students 24

Table 2. Demographic Summary of the Students In Terms of Groups 41

Table 3.Gender 41

Table 4. Mathematics GPA Level 41

Table 5. Students' Pre-Course Programming Language and Tools 42

Table 6. Activity Plan for the Groups 49

Table 7. Item/Factor Loads for MAS 52

Table 8. Item/Factor Loads of for CTSSES 54

Table 9. Paired Groups t-Test Differences Between Mean Values from MAS Scores 69

Table 10. Differences Between Participants' Opinions based on MAS Scores (by Group) 74

Table 11. Covariance Analysis of MAS Pretest–Posttest Scores 75

Table 12. Effect of Group Applications on Pretest Scores 76

Table 13. Effect of Group Activities on Posttest Scores of MAS 76

Table 14. Effect of Group Applications and Pretest on MAS Posttest Scores 77

Table 15. Sobel Analysis Results 77

Table 16. Effect, Direct Effect, and Indirect Effect of Group Applications on MAS Posttest Scores 78

Table 17. Mean Scores for Paired Groups t-Test of Students' CTSSES (Pretest–Posttest) 79

Table 18. Between Participants' Opinions Regarding Subscales of CTSSES Scores (by Group) 83

Table 19. Covariance Analysis CTSSES Pretest–Posttest Scores 84

Table 20. Group t-Test: Difference between Students' CPSES Scores (Pretest–Posttest) 86

Table 21. Participants' Opinions Regarding CPSES Scores (by Group) 89

Table 22. Covariance Analysis of Students' CPSES Pretest-Posttest Scores 90

Table 23. Paired Groups t-Test: Differences Between Mean Scores from Pretest & Posttest of SBPAT 92

Table 24. Differences Between Participants' SBPAT Scores (by Group) 95

Table 25. Details and Assigned Codes for Interviewed Students 98

Table 26. Teachers' Views on Programming Teaching 101

Table 27. Codes Created for Students' Negative Opinions about Small Basic 108

Table 28. Codes Created for Students' Positive Opinions about Small Basic 109

Table 29. Codes for Students' Difficulties in Using Small Basic 111

Table 30. Codes Created for Factors That Helped Students in Learning to Use Small Basic 113

Table 31. Codes Created for Students Learning Programming Using Small Basic 115

LIST OF FIGURES

FIGURES

Figure 1. Components of CTS (based on Wing, 2006) 11

Figure 2. Reasons to Teach Programming to Young Learners (based on Guzdial, 2015) 19

Figure 3. Extreme Values Obtained from Mathematics Attitude Scores 73

Figure 4. Outliers of Mathematical Attitude Pretest–Posttest Scores 73

Figure 5. Conceptual Diagram of Multiple Regression Analysis 75

Figure 6. Extreme Values from Computational Thinking Skills Self-Efficacy Scale 82

Figure 7. Outliers of Computational Thinking Skills Self-Efficacy Scale (Pretest–Posttest) 83

Figure 8. Extreme Values from Computer Programming Self-Efficacy Scale 88

Figure 9. Outliers of Computer Programming Self-Efficacy Scale Pretest–Posttest Applications 88

Figure 10. Extreme Values from Small Basic Programming Achievement Tests .. 91

Figure 11. Outliers of Small Basic Programming Achievement Test (Pretest–Posttest) 92

Figure 12. Differences Between Groups for Students' Small Basic Programming Achievement Test Pretest–Posttest Scores 94

Figure 13 Differences Between Groups for Students' Small Basic Programming Achievement (Pretest–Posttest) Scores 95

Figure 14. *Themes and Sub-themes of the Study* 97

Figure 15. Similarities Between Students' Views (by Group) 99

LIST OF ABBREVIATIONS

ABBREVIATIONS	
CPSES	Computer Programming Self-Efficacy Scale
MAS	Mathematics Attitude Scale
CTSSES	Computational Thinking Skills Self-Efficacy Scale
SBPAT	Small Basic Proramming Achievement Test
CTS	Computational Thinking Skill
CS	Computer Science
SB	Small Basic
ICT	Information and Computer Technology
IA	Interdisciplinary Approach
TA	Traditional Approach

CHAPTER 1

INTRODUCTION

Educational technology has become a key element in today's education due to the necessities of dealing with the COVID-19 pandemic. Education systems worldwide have had to abruptly change and this change has significantly increased the importance of using educational technology, with teachers at all levels pushed to their limits to adopt this new way of teaching. This new dimension raises the concept of using technology and understanding the machine world not only in higher education, but also at the K-12 school-age education level too. Due to the requirements for online teaching of school-aged children during the pandemic, the whole area of online teaching utilizing education technology has become an essential skill for all. In order to understand today's new world and technology-enhanced learning, introducing computer science to learners at an early age is considered to be vital. Therefore, the requirement for teaching programming to young students has expanded significantly, with technology education now focused more on programming at an early age as an important element in the development of children's higher-order and problem-solving skills (Dagdilelis et al., 2004). Brennan (2017) described programming as specified codes that enable users to control the activities of a computer. Educational technology literature highlights that programming enables students to examine the nature of a problem, the structure of its design, and the function of significant knowledge through the combined skills of computational thinking and higher-order problem-solving. As programming requires a significant level of knowledge and skills, methods of teaching are categorized in terms of the environment, knowledge of the domain, and the choice of tools that are employed for a specific task. These categories are significant to making the teaching of the subject both effective and efficient, and to facilitating the learning of programming (Smith et al., 1997).

Various scholars have attempted to find different activities to teach programming to pupils in the easiest way possible. Worldwide initiatives through stakeholders such as the Computer Science Teachers Association (CSTA), the International Society for Technology in Education (ISTE), and the Association for Computing Machinery (ACM) have seen computer science standards added to curricula, not only in higher education, but also at all levels of K-12 instruction too. CSTA's K-12 Computer Science Standards highlights the significant influence of programming on computational thinking, which helps students conceptualize a problem better and allows them to select the best strategy and tool to improve their problem-solving skills (Seehorn et al., 2013).

Programming education improves not only problem-solving skills, but also computational thinking skills as well. Researchers have stated that everyone should learn programming in order to gain a level of new skills for today's world; providing individuals with significant life-skills through the learning of programming enables individuals to interact with computers, and to try to understand its concepts by reading, writing, and debugging what is essentially computer-based code across all areas of daily life (Duncan et al., 2014; Guzdial & DiSalvo, 2013; Kalelioğlu & Gülbahar, 2014; Resnick & Siegel, 2015; Thies & Vahrenhold, 2013; Tsan et al., 2016). Wing (2006) indicated that computational thinking is not just the principle of thinking and acting like a computer, but is far more than that by including feelings and senses in relation to programming. Therefore, improving the programming skills of students also enables them to improve their computational thinking skills.

The ISTE, CSTA, and ACM standards all state that providing interdisciplinary examples to students during the learning of programming is considered important in order to enlighten students about their future career choices, because it enables them to create innovations and applications in numerous different areas, which may in turn affect their choice of career (International Society for Technology in Education & Computer Science Teachers Association, 2011). Therefore, the role of both computational thinking and interdisciplinary collaboration is seen as significant to the learning of programming.

Maloney et al., 2008 stated that all disciplines can benefit from computer science, since it is determined that other disciplines are connected with computer science, yet, it is important to have a good all-round education so as to provide learners with an efficient environment which requires significant skills of collaborative working and computational thinking. Scholars have stated that both the teaching and learning of programming is not considered to be easy as it requires a deep understanding of the problem in question, and solutions that follow a correct algorithmic course (Esteves et al., 2008; Guzdial, 2004; Kelleher & Pausch, 2007; Maloney et al., 2008; Resnick et al., 2009; Saeli et al., 2011). De Jesus Gomes et al. (2015) highlighted that learning programming is seen as difficult because of the teaching method applied. Integrating an interdisciplinary approach and real-life-based examples can therefore facilitate the learning of programming, and thereby maximize learning retention in students.

1.1 Problem Statement

The ubiquitous presence of technology in modern daily life has become significant for today's younger generation, making them curious about computers and their environment by understanding the way in which these machines actually work. Angeli et al. (2016) indicated that the economic rationale is led by computer science, requiring today's employees to understand how to make machines communicate through programming, and that this requires computer science to be tackled within early age education, with curricula that includes new content in order to make the subject easier for young minds to comprehend.

Programming is defined as the process of developing the source code of a device, and then implementing the developed code to enable a device to undertake a prescribed task (Kalelioğlu, 2015; Rodríguez et al., 2017; Seng & Yatim, 2014; Tan et al., 2014). Since computer technology is still expanding exponentially, its effect on education and computational thinking has naturally increased significantly too. Teaching programming is not considered to be easy because it requires the combined skills of problem solving, critical thinking, creative thinking, and algorithmic thinking, and that it is seen as much harder when it comes to teaching these skills to

younger students (Becker, 1993; Kelleher & Pausch, 2007; Mészárosová, 2015; Miliszewska & Tan, 2007; Saeli et al., 2011). In order to facilitate the learning of programming at an early age, educators seek out new methods and the most appropriate programming applications.

The teaching of a programming language using traditional approach focuses more on structure, syntax, and in the debugging of code. However, through such traditional methods, students can only practice programming on small-scale problems, and on specific examples in order to build a solution for a given problem, and thereby lacks real-life-based examples in solving problems through the application of a programming language. Previous researchers have stated that students do not possess adequate skills to apply real-life contexts in their solutions, due in part to their own lack of life experience and in real-life-based scenarios applied in the instruction they receive (Jenkins, 2002; Lin et al., 2019, Özmen & Altun, 2014; Sins et al., 2005; Wing, 2006). Studies have shown that learning a programming language is more about facilitation when it comes to different notations (Atmatzidou & Demetriadis, 2016; Saeli et al., 2011); hence, the biggest challenge in computer science education programs is not just in terms of teaching methods, but in the learning of programming. Accordingly, decades of research have stated that the major mistakes encountered with the teaching of programming have been in not seeing problems as a composition taken from real-life experience, and that whilst students know what language to use, they do not know how to put it all together when attempting to solve a problem due to a lack of concrete examples applied during their programming training (Caspersen & Kolling, 2009; Oliveira Aureliano, 2013; Sins et al., 2005).

Within the rapid growth in technology seen in recent years, the importance of programming education at early age has been recognized as significant for the future job market and for the next generation's career plans. This was felt more during the COVID-19 pandemic period. Draganoiu et al. (2017) stated that the learning of programming requires a significant level of problem-solving skill, because it activates higher-order thinking skills that programmers need in order to solve problems easier. In transferring abstract problems to the physical world, a second discipline is needed. Interdisciplinary teaching and learning involves collaborative

efforts that encompass multiple disciplines in attempting to understanding a problem (Cassel, 2011; Mahadev & Connor, 2014). Since mathematics plays a significant role in computer science, the combining of both programming and mathematics creates a natural link between real-life problems and the ability and means to solving these problems through the application of computer programming.

1.2 Significance of the Study

Within the current growth in technology, the needs of today's employers for employees who know how to program have increased. Hence, the learning of programming is becoming not just popular at the higher education level in terms of career choices, but for programming education at early age too in preparing today's students for the realities of tomorrow. Researchers have stated that knowing how to program is considered a requirement as part of 21st-century skills which support problem solving, algorithmic thinking, and computational thinking (Guzdial & DiSalvo, 2013; Kafai & Burke, 2014; Saeli et al., 2011; Tsan et al., 2016; Wing, 2006).

The purpose of the current study is, therefore, to investigate the application of mathematics as a secondary discipline in the teaching of programming to students at a young age. This study aims to examine the influence of mathematics on computational thinking and programming skills through an interdisciplinary collaboration as part of learning programming. The study also aims to present how a valuable collection of computer science education is important in secondary school education by promoting interdisciplinary collaborative working, with activities based on computational thinking. In a broader context, the study is aimed at benefitting educators who teach programming at the secondary school level by providing them with example lesson plans prepared not just according to traditional approach of programming instruction, but based on the collaboration of mathematics as an interdisciplinary subject.

1.3 Research Question

The current study is based on a primary research question, with four qualitative plus two quantitative sub-questions. These primary and sub-research questions are as follows:

1. What are the effects of teaching programming with a traditional approach (TA) and an interdisciplinary approach (IA) on secondary school students' attitudes, computational thinking skills, programming skills, and academic achievement level?

    a. Is there a significant difference between the TA and IA groups according to their Mathematics Attitude Scale scores (MAS)?

    b. Is there a significant difference between the TA and IA groups according to their Computational Thinking Skills Self-Efficacy Scale (CTSSES) scores?

    c. Is there a significant difference between the TA and IA groups according to their Computer Programming Self-Efficacy Scale (CPSES) scores?

    d. Is there a significant difference between the TA and IA groups in their retention of Small Basic Programming Achievement Test (SBPAT) results?

    e. What do teachers think about the teaching of visual programming tool using TA and IA methods?

    f. What do students think about visual programming tool and its processes?

1.4 Organization of the Study

In the first chapter of the study the problem statement, the purpose of the study, research questions, significance of the study, and the definition of terms used in the thesis are addressed. The study continues with Chapter Two, which provides a literature review of interdisciplinary collaboration, computational thinking skills, programming education, and how these are taught at an early age. Chapter Three focuses on the methodology of the study, and provides detail about both the pilot and main parts of the study. This includes the research design, the instruments used in the study, the data analysis, and the methods applied to ensure the study's reliability and validity. Chapter Four details the findings of the study according to the research questions, whilst the results are discussed in Chapter Five, along with the author's conclusions.

CHAPTER 2

LITERATURE REVIEW

2.1 What is Computational Thinking Skill?

The rising trend in improving the 21st-century skills of students through education includes the skill of thinking like a computer. Thinking in the way that a computer operates is termed as “Computational Thinking,” which is a term recognized since the 1950s, along with the skills-based concept of problem solving (Denning, 2009; Haseski et al., 2018; International Society for Technology in Education & Computer Science Teachers Association, 2011; Kalelioğlu et al., 2016). Even though there are terms that define computational thinking, its meaning covers more than just thinking in ways that a computer works. Beecher (2017) stated that defining computational thinking is challenging, not because it includes a wide range of both abstract and concrete principles.

Researchers have defined computational thinking as: a process which involves an arriving at a solution for a given problem (Wing, 2008); a mental activity that examines a problem and develops an automated solution (Yadav et al., 2017); the mental process where problems are formulated with inputs and outputs through the performing of an algorithm (Denning, 2009); and using computation for the purposes of problem solving, as well as to explore and make new discoveries (Hemmendinger, 2010). On the other hand, computational thinking has been defined as a function of the brain in terms of setting problems and formulating appropriate solutions, understanding human behavior, and seeing problems in such a way whereby computers can deliver the solution (Berry, 2015; Grover et al., 2014; Yadav et al., 2014). The International Society for Technology in Education and the Computer Science Teachers Association (2011) jointly stated that computational thinking can

be defined as formulating problems to use tools such as computers for identifying, analyzing and representing data. From the similarities of these various definitive statements, computational thinking can be defined in general as the ability to solve a problem by utilizing an abstract, focusing on algorithmic design, and applying a solution in collaboration with real-life activities.

Computational thinking is a method of solving problems that cannot be achieved through just one singular outcome. Thus, computational thinking is a combination that resolves a series of problems through prediction and abstraction (Baytak & Land, 2011; Grover & Pea, 2013; Kafai & Burke, 2014; Sengupta et al., 2013; Wing, 2006). Even though programming seems an essential element for the improvement of computational thinking skills, Grover and Pea (2013) stated that programming was not the only means to improving computational thinking, as it also requires the demonstration of activities which support higher-order thinking skills.

2.1.1 Characteristics of Computational Thinking

According to Selby and Woollard (2013), computational thinking encompasses two important thinking skills areas, which are "logical thinking" and "algorithmic thinking." In terms of problem solving as a skills area, the relevant terms are "problem solving" itself, as well as "analysis" and also "generalization." For computer science concepts, Selby stated the specific terms as being "systems design," "automation," and "more general computer science concepts." As to the concept of imitation or representation, the specific terms that Selby put forward were "modeling," "simulation," and "visualization."

Researchers appear to agree that computational thinking skills can be improved not just through programming and problem solving, but with visualization, and through integrating problems that are based on real life, the active debugging of a programing errors, and through abstraction (Armoni, 2011; Baytak & Land, 2011; Brennan & Resnick, 2012; Kwon et al., 2012; Maloney et al., 2010; Selby, 2013; Wing, 2008).

Wing (2006) put forward that the components of computational thinking skills are as illustrated in Figure 1.

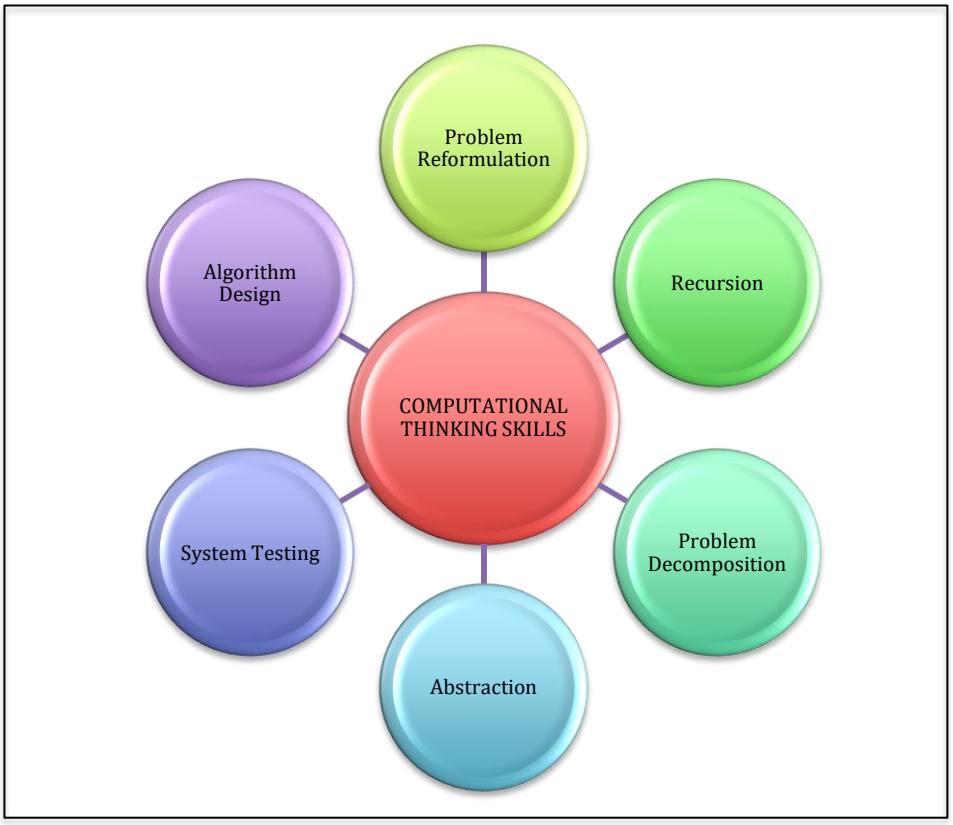


Figure 1. Components of CTS (based on Wing, 2006)

Shute et al. (2017) then went on to further define Wing's computational thinking skills components as follows;

- Problem reformulation: Rebuilding a problem into a new one in order to solve it easier,
- Recursion: Building a new set based on previous knowledge and information,
- Problem decomposition: Splitting the problem into smaller pieces so as to manage it easier,
- Abstraction: Shaping the main faces of a problem,

- Systematic testing: Determining actions to solve problems in logical aspects.
- Algorith design: Developing a step by step solution

Various studies have highlighted that what makes computational thinking different from other forms of thinking such as engineering, systems, and mathematics, is that computational thinking helps people to better understand a problem, without physical limitation, through the creation of a model based on abstraction (Razzouk & Shute, 2012; Shute et al., 2010, 2017). Therefore, scholars agree that computational thinking is not being able to think like a computer, but involves at least six components of solving problems through the integration of cognitive processes (Razzouk & Shute, 2012; Shute et al., 2010, 2017; Wing, 2006; Maloney et al., 2010; Selby, 2013). Since computational thinking focuses on analyzing the whole system, as a way of thinking how to facilitate problem solving more creatively, and is deemed to not be limited by certain conditions, it can be said that computational thinking is an overarching concept that covers not just algorithmic thinking and problem solving, but also mathematical thinking, systems thinking, and engineering too (Shute et al., 2017).

2.2 Computer Programming Education

Programming education has been trending rapidly worldwide due to the associated positive effect on both the global economy, and also as a result of new and improved technological advances. Programming refers to the writing of a set or series of instructions and commands that a computer comprehends for the operation of a specific task. Developing software is divided into two categories; from the perspective of engineers or technicians, and from the educational perspective according to its integration into modern-day pedagogy.

The engineering perspective's focus is on professional software development, which usually requires a systematic work plan with a team of engineers; whereas, the

educational perspective focuses on individual development that is based on mental models and recognized teaching methods integrated within an established pedagogical approach (Robins et al., 2003).

Programming has been considered as one of the most compelling components of today's educational programs, especially within K-12 education. In other words, people currently live in a digital era whereby computers are ubiquitous and represent what has become an essential part of daily human life (Tsai et al., 2019). Every pupil or student is entitled to some form of technological access, whether in the form of laptop or desktop computers, or through the use of mobile solutions delivered to tablet computers and smartphones. As a result, almost all solutions today have call for some form of programming. However, the teaching of programming in schools needs to be undertaken in accordance with the appropriate pedagogical perspective and approach, and both educational designers as well as instructors and teachers should be fully aware of the pedagogical concepts according to the developmental stages of their target student group.

Scholars mostly agree that every single person should be aware of the importance of programming, with many new careers now either directly or indirectly related to the computer sciences (Duncan et al., 2014; Fojtik, 2014; Guzdial & DiSalvo, 2013; Oliveira Aureliano, 2013). Hence, in a short film directed by Chilcott (2013), both Apple Inc.'s late CEO Steve Jobs and Microsoft Corporation owner Bill Gates agreed that everyone should learn how to compute a program as it teaches them how to think, and that some form of basic code development provides learners with the primary computing skills they require for almost any career of their choice in the future. It is important to understand too, as Zhang (2017) insisted, that K-12 education systems are based on the "learn by doing" approach. This includes the application of step-by-step instruction, which guides learners through the various aspects of programming. As such, BASIC (Beginners' All-purpose Symbolic Instruction Code) can be described as an entry-level approach as the first step in computer programming (Harvie et al., 2018). As an initiating point, learners learn to

understand and appreciate programming through early steps that describe the subject in easy, plain language. On the other hand, individuals should also learn to program in order to improve their abilities in numerous other skills areas such as higher-order thinking, critical thinking, creative thinking, and problem solving (Kalelioğlu & Gülbahar, 2014).

Besides its effect on students' academic ability, programming helps them progress in life, in a world hugely affected and influenced by endless new technological innovation. Sáez-López et al. (2016) stated that programming has been integrated throughout the K-12 curriculum (i.e., primary, secondary, and high school) in countries such as Estonia, the United Kingdom, Sweden, and in Finland. Until recently, programming in Turkey was mostly associated with graduate level education. However, based on the new trends of the last decade, and with increased popularity of events such as "Hour of Code," "Code Week," "FIRST Lego League," "World Robot Olympiad," and the "Bebras Challenges," the learning of programming in Turkey has now been made available to children as from the age of 10 years old. The reason that programming has been integrated into the Turkish educational system is to help youngsters to become less afraid of computer science, and to bring their attention to the importance of programming in today's world from an early age (Karabak & Güneş, 2013).

The acquisition of computer programming knowledge among K-12 students fosters a general broadening of skill as well as increasing overall educational understanding. This knowledge is deemed to be important in shaping the students' aspirations and to their daily thoughts on making the world a better place for them individually, for their family and friends, and for others too. Swacha and Baszuro (2013) stated that the learning of programming skills can change the minds of many students in positive terms with regards to the careers they may pursue later on in life. Ideally, such knowledge broadens their understanding and enhances their IQ as well. Skills associated with computer programming also help students to more easily complete

simple day-to-day activities such as word processing, email correspondence, online educational studies, and conducting Internet-based research.

As students' progress through to the higher stages of the K-12 education system, they should be able to describe the various challenges that can be solved through programming. Students are then required to describe the challenges in a format that can be translated into code, and then emulate the code that they need to come up with. Thereby, as learners increase their knowledge within the K-12 education ladder, they should be able to understand much of what is incorporated in coding, including control structures, variables, methods, and input files, etc. (Swacha & Baszuro, 2013). Building confidence and choosing the correct tools are also seen as essential aspects of teaching programming among K-12 students. During the earlier stages, learners should be introduced to easy-to-use tools such as *Thonny* for python IDE and *repl* for online IDE, since neither require any form of installation nor rely on in-depth instruction. On the other hand, the right types of tools should be used in order to promote confidence through usage (Murata & Kakeshita, 2016). This approach should be taken in steps so as to promote easy understanding among young learners and to foster their skills for future induction to the complexities of computer programming.

Guzdial and Morrison (2016) stated that a variety of people who learn how to program become interested in the area because of the level of growth seen in the software industry. Whilst most programming platforms are comprehensible to students at the undergraduate level, teaching programming learning to younger aged children can be challenging for teachers. The following subsections outline programming education for early ages, the relationship between programming and computational thinking, and the programming tools used by students at the K-12 level.

2.2.1 Programming Education for Early Ages

Both teaching and learning programming has been said to be difficult due to the requirements of the skills involved (Bennedsen & Caspersen, 2007; Bornat & Dehnadi, 2008; Guzdial & Forte, 2005; Jenkins, 2002; Mazman & Altun, 2013; Özmen & Altun, 2014). Thus, teachers seek new ways to teach programming that are easier, not just for young students but also for undergraduates as well. De Raadt et al. (2002) stated that it is difficult to learn how to program, hence teachers are looking for new methods to facilitate the learning of programming through motivating and supporting students according to their needs and learning styles. Studies have indicated that in order to help undergraduate students struggle less with programming, the fundamental logic involved should be taught at a much earlier age (Guzdial & Forte, 2005; Karabak & Güneş, 2013; Malan & Leitner, 2007; Resnick et al., 2009; Vatansever & Baltacı Göktalay, 2018).

Learning programming at an early age enables students to learn how to follow algorithms, to be more creative in their thinking by reflecting problems in different ways, and in developing and applying higher-order skills to describe their direction of thought whilst solving problems (Clements & Gullo, 1984; Gorman & Bourne, 1983; Grover & Pea, 2013; Karabak & Güneş, 2013). Since the importance of learning programming at an early age has become more popular, school curricula related to programming has begun to be developed and introduced at the K-12 level. Besides countries such as Sweden, Austria, the Netherlands, the United Kingdom, Australia, Germany, and the United States, Turkey has also introduced programming into its K-12 curricula as a means to raising its importance (Battal & Tokel, 2020; Gal-Ezer & Stephenson, 2014; Kalelioğlu & Gülbahar, 2014; Sáez-López et al., 2016; Yadav et al., 2017).

Programming influences society, and today's youth are starting to be prepared to face the modern technologies of the future by learning programming through easier methods. Programming is present in every contemporary field of life, and should

therefore be learned at an early age in order to help children better understand mathematics and science, to be able to command higher-order problem-solving skills, and to be better able to deal with understanding the world in which we live, as well as that of tomorrow. Programming establishes a new sense in students, enabling them to establish links between different salient disciplines (Saeli et al., 2011). In addition, Kelleher and Pausch (2007) argued that programming can be learned at any age if the mechanics of the programming taught are simple to learn and to use, and that feedback is given and adequate support provided to learners in order to increase their motivation to learn programming. Programming is also important because it provides the means for programmers to make machines "talk," which helps others to observe how computers process the commands they receive. Berry (2015) wrote in the "QuickStart: Computing Primary Handbook" that programming presents a great chance to test out our ideas with immediate feedback. Thus, programming helps children to think with higher-order skills, set an idea in play, apply it in order to test it, and further develop it according to the feedback received, which represents an opportunity to increase learning motivation through the visualization of a product as an outcome.

Teaching programming to today's young students requires being aware of the latest curricula changes, including ACM's K-12 curriculum, ISTE's Computer Science Standards, and CSTA's K-12 model, and requires knowing how to promote programming so as to attract the attention of students by teaching it creatively (Porter et al., 2013; Saeli et al., 2011; Ward & Parr, 2010; Winslow, 1996). Holmboe et al. (2001) also stated that teaching should not just be about the transference of computer science technical knowledge to students, but that teachers should be aware of the required pedagogical content knowledge whilst teaching programming to their students. Thus, when secondary school students are taught how to program, besides gaining the requisite technical knowledge, they do so in collaboration with other disciplines. Saeli et al. (2011) claimed that teaching programming is not just about the inclusion of informatics in lesson plans, but linking informatics with

mathematics, physics, and also engineering, which are each influential to the learning of computational thinking skills.

Therefore, the questions to be addressed are, how can pedagogy be included in programming courses, and what ways can teaching programming be effectively and efficiently conducted with secondary school students? Grover et al. (2014) stated that "CS unplugged" (Computer Science Unplugged; teaching material based on engaging games and puzzles) should be used while introducing programming to young students, because it provides significant activities to help students learn about the nature of computer science. On the other hand, Kelleher and Pausch (2007) stated that storytelling-based projects can make the learning of programming much easier because it engages students through motivation and creative thinking. Thus, studies in the literature have shown how both CS unplugged and storytelling-based activities can enhance the motivation of students towards the learning of programming (Burke & Kafai, 2010; Feaster et al., 2011; Taub et al., 2012; Theis & Vahrenhold, 2013; Werner et al., 2012).

Even though programming is suggested to be taught at an early age, it is argued, however, that the aim of professional programmers and K-12 educators are quite different. The gap regarding how to teach programming to younger students compared to adult professionals has been steadily growing, with scholars still looking for the best methods to facilitate the learning of programming. Battal (2018) stated that educators should collaborate both with computer scientists and also with teachers from other disciplines in order to address this gap as interdisciplinary collaboration has been suggested as a means to making programming easier to learn. But why is it considered so important to learn programming at an early age? Guzdial (2015) developed six important ideals as to why teaching programming to young students is important, as shown in Figure 2.

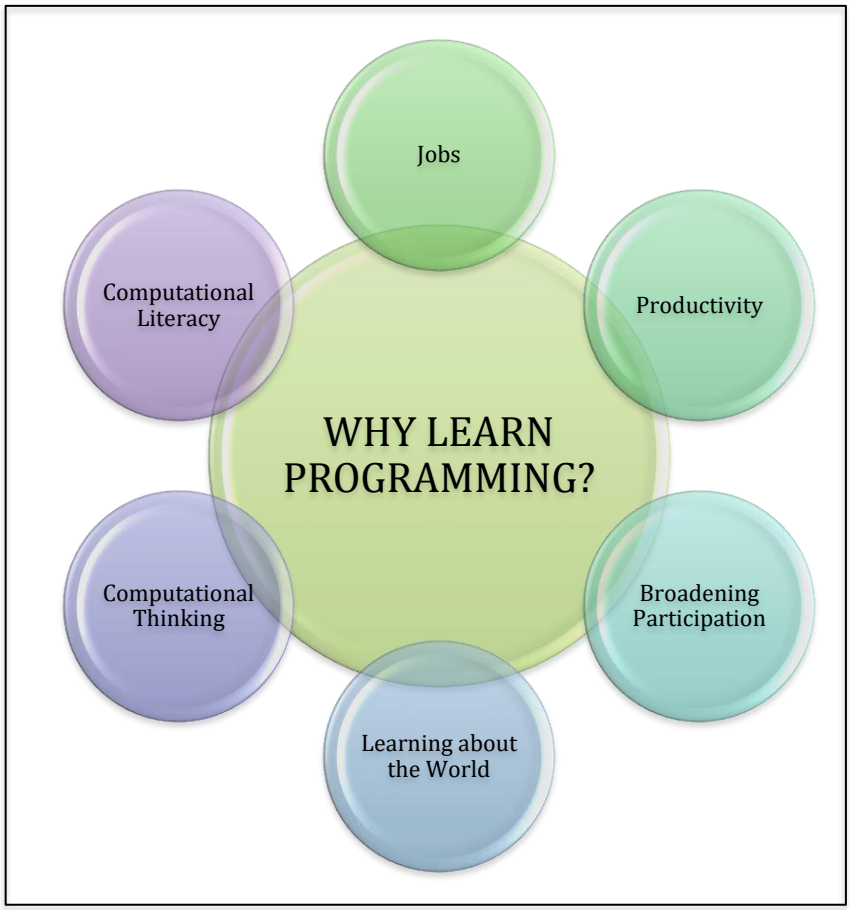


Figure 2. Reasons to Teach Programming to Young Learners (based on Guzdial, 2015)

In terms of gaining employment, Guzdial (2015) highlights that the growth seen in the field of computer-based technology requires highly skilled employees in the field of computer science, and that new careers will be based on computing knowledge. The second reason mentioned was with regard to "learning about the world." Computing is now part of our lives and, because we live in a computational world, students must learn and understand computing in order to communicate in the language of the era. Third, learning programming increases the ability of computational thinking, as it links problem solving and creative thinking. Computational literacy was stated as the fourth goal of learning programming, since computers and technology are the new media; therefore, to understand its literacy is one of the fundamental requirements of familiarization with the domain of modern computing. The next goal that Guzdial (2015) mentioned was the importance of

productivity. In order to fabricate products related to computers and to find employment in this new career area, learning programming has become an essential step as people not just use computers, they are starting to think about the functionality in order to achieve much more than just computing. The final reason Guzdial put forward as to why programming should be learned is to broaden participation. Learning programming should be for everyone and everyone should be treated equally without discrimination.

2.2.2 The Relation between Computational Thinking and Computer Science

Computational thinking requires understanding the concepts that offer knowledge to the community. Learning programming languages is one aspect that is used to ensure that positive effects can be achieved for the community, and what follows are the effects of that learning. One of the impacts of learning programming is that it provides knowledge to design systems that are used in today's business world. Another result is that through learning to program with computational skills, individuals can understand the various behaviors portrayed by others in the community (Howland et al., 2009). Programming learning is also essential to the community as it helps learners to solve problems in society. Individuals who learn computational languages gain a wider knowledge that is used to interact with computer-related systems. These impacts therefore profoundly contribute to their computational thinking and academic growth.

The concepts of learning programming and computer skills are highly interrelated. Computer skills are those that help individuals to understand the learning of programming. Also, through learning to program, students can learn a high level of computer skills that can be utilized in solving real-life problems. These two interact in order to help in the growth of students who learn these skills (Cansu & Cansu, 2019). According to Wing (2006), although computational thinking and computer science are two different disciplines, students can gain the opportunity to improve

their computational thinking skills whilst learning to program. Also, students learn to think based on the computational skills gained in their learning. Learning programming depends on the knowledge of computational skills, which are highly significant to the learning of programming. The two aspects therefore require each other in order for the educational objectives to be achieved.

Studies have indicated that computer science enables people to utilize their skills of algorithmic thinking, critical thinking, and problem solving, which are each concepts used in the transference of computational thinking into real-life solutions (Aho, 2012; Czerkawski & Lyman, 2015; Ioannidou et al., 2011; Israel et al., 2015; Wing, 2006). Grover and Pea (2013) stated that computational thinking focuses on decaying a problem and then generalizing it into patterns; which of course is just the same as applied in computer science, whereby programmers correct coding errors through debugging and recursive thinking. Computer science promotes computational thinking skills because both have the same primary aims, which are problem solving, thinking critically, and thinking creatively. Features related to the learning of programming that involve computational skills require that programmers are in a position to define specific and precise instructions. First, step-by-step instructions are written that can then be used as the basis for designing a program. The program structure is then developed as applicable in order to solve the issues presented in the professional dimension. Another feature is the ability to project a system constituted of different elements that defines the areas that need to be addressed in the program. These components of the program vary, but should not overlap in their functionality (Kurilovas & Dagiene, 2016). Systems should therefore be easy to understand, and thereby function as would a computer in the eyes of the user. The other element of learning programming is the ability to understand complex systems of behaviors, as portrayed through the interactions of computational skills and programming language learning. Also, the programmer should have the ability to design a system that reveals operations and information related to the original problem or task. In general, computational thinking is the ability to organize and analyze data logically.

Once a problem has been identified and appropriately analyzed, then, potential solutions may be put forward in order to solve the problem.

Barr and Stephenson (2011) highlighted that computer science has an important role to play in the wider community, because it improves a person's algorithmic thinking skills through problem solving and it helps communities to integrate computational thinking methods automatically based on natural interdisciplinary relations. Also, Hemmendinger (2010) indicated that computer science and computational thinking share many disciplines such as constructing, modeling, and debugging errors, as well as analyzing them through a constructive and organized approach. Thus, when an instructor sets a course in computer science, they will automatically design a course within which computational thinking is integrated.

Lu and Fletcher (2009) stated in their research that computational thinking in terms of its pedagogical approach is related to teaching the basics of algorithmic thinking, because the focus is all about reading and writing the individual steps of a problem in order to solve it critically and coherently. Researchers have focused, therefore, on how to improve different skills' fields through the integration of interdisciplinary collaboration. According to Çınar and Tüzün (2017), disciplines that involve robotics, programming-based building bricks, and digital gaming activates learners' computational thinking skills. Therefore, the role of programming in terms of computational thinking cannot be denied, because it both serves the building of 21st-century skills (Alsancak Sirakaya, 2019) and algorithmic thinking skills in terms of debugging, problem solving, as well as simulation and visualization (Kazimoglu et al., 2012).

Computational thinking is considered to work well at the K-12 level in ensuring that certain pedagogical objectives are met. Instructors who teach computer-based thinking have introduced certain changes to K-12 curricula as a means to contributing to the students' understanding of these concepts. At this level, exams are aimed at ensuring students understand the ideas behind computer-based thinking (Sands et al., 2018). Information technology as a domain significantly requires that

computational thinking skills are developed throughout K-12 education in order to realize the expected levels that today's employers demand in terms of achievement in computational skills. K-12 education is therefore utilized so as to extend students' knowledge about computers based on established modeling phenomena, and the designing of efficient and effective algorithms. The K-1 school year is considered vital in terms of the learning of initial computational skills and, therefore, necessary changes to the curricula should be implemented in the computer sciences area to enhance learning in young students.

2.2.3 Programming Tools for K-12 Students

Having the title of computer programmer has become ever more popular due to the technological growth seen in recent times. Parallel to this increased popularity, the development of new programming tools has also experienced significant growth too, and especially so in the field of educational tools. Numerous tools now exist that enable students, both young and adult, to learn how to program easily due to their user-friendly interfaces. Scholars have stated that programming learners include a wide age range due to the varied career positions now available based on computer technology. This learners' age range now starts from kindergarten, and progresses right through to higher and graduate education (Guzdial, 2015; Kalelioğlu & Gülbahar, 2014; Karabak & Güneş, 2013; Vatansever & Baltacı Göktalay, 2018). Examples of some of the programming tools now used in K-12 education are provided in Table 1.

Table 1. Programming Tools for K-12 Students

Level	Programming tool
Kindergarten	Code.org, Scratch Jr.
Elementary school	Kodable, Scratch, Tynker, SpriteBox Coding, Code.org
Secondary school	Kodu Game Lab, Small Basic, App Inventor, Alice, Code.org, LEGO MindStorms, O-Bot, mBlock, Minecraft for Education, Hopscotch, Codemonkey
High school	Python, CoSpaces Edu, Arduino, KOOV

Code.org

The Code.org platform was launched in 2013, and was aimed at teaching younger learners how to program through simple "drag and drop" programming. The advantages of this type of tool are its widespread availability, now offered in 56 languages, and its design which is based on popular games or movies such as Angry Birds, MineCraft, Star Wars, Frozen, etc., which naturally make the product attractive to young students. The CEO of Code.org, Hadi Partovi, established the non-profit educational organization for the purposes of teaching algorithm, conditions, and the fundamentals of programming to children in a fun way. Support for the platform has been garnered from politicians such as former US President Barack Obama, technology idols such as Microsoft Corporation's CEO Bill Gates, Facebook Inc.'s CEO Mark Zuckerberg, DropBox Inc.'s CEO Drew Houston, and Cofounder of the Valve Corporation Gabe Logan Newell. Partovi's Hour of Code activities have also received widespread support from celebrities such as Shakira, Ashton Kutcher, will.i.am, and Chris Bosh.

Scratch Jr. and Scratch

Both Scratch Jr., which was developed for children aged 5 to 7 years old, and Scratch, aimed at children aged 8 to 16 years old, are free programming languages developed

by the Massachusetts Institute of Technology (MIT) that aim to teach children programming through interactive stories, animations, and games using the "block-coding" method. The visual environments of Scratch provide young programming learners with the opportunity to explore and develop basic programming sets. Good (2011) stated that the most clear benefit of Scratch is that it does not require any prior knowledge about programming, due to its use of the drag and drop programming method. Both of the Scratch tools are now in common use across more than 150 countries worldwide, providing access in more than 40 languages and therefore a considerable advantage for many new learners. Scratch has been shown to have a positive effect on the computational thinking skills of its users; largely due to the founding concept and its development having been based on the LOGO project, which included a constructivist theoretical approach in that learners learn by themselves (Sáez-López et al., 2016).

Tynker

Tynker for schools was launched in 2013, and was aimed at teaching children to create games and programs by dragging and dropping code blocks. Its design and basic principles are based on Scratch, with the only difference being that Scratch is free and open-source, whilst Tynker is a commercial tool. It includes different types of courses for coding such as simple coding, game design, and even robotics. Also, it follows a hierarchical design according to the different ages and development levels of its users. For example, children in the 5 to 7 years old age group learn to solve logic problems and create simple applications, whilst 7 to 13 year olds construct games and explore the multidisciplinary STEM subject. For users aged over 13 years old, they learn HTML, CSS, JavaScript, and Python, which are more advanced programming and script languages. Tynker offers users self-paced learning that is therefore appropriate for all different levels. It can also be used on different media such the web browser of a desktop or laptop personal computer, or by using a mobile application on a tablet computer or smartphone. The tool received financial support from both angel investors (individuals) and institutional investors. Also

many brands work in cooperation with Tynker, including Google, Lego, Microsoft, Minecraft, Facebook, and Apple, etc.

Microsoft Small Basic

Small Basic version 1.1 which is a programming language associated with the "integrated development environment" (IDE) was released by Microsoft, in 2015.

> *Microsoft Small Basic is a simple BASIC programming environment designed specifically for kids to help prepare them for more complex programming languages like Visual Basic, Visual C# and Java. Small Basic also includes a 'graduate code' button which automatically converts the student's source code into Microsoft Visual Basic.* (Conrod & Tylee, 2013)

BASIC here stands for Beginner's All-purpose Symbolic Instruction Code. The aim of Small Basic is the provision of a programming language with an easy and user-friendly interface that uses 14 keywords for beginners during their transition from block-based coding to text-based coding. As such, it forces students to employ higher-order thinking skills because of the method set as text-based coding. Students can advance using mathematics libraries that are embedded into Small Basic in order to facilitate the coding of functions. Also, it supports basic data types such as strings, integers, and decimals.

App Inventor

Developed by Google in 2010, App Inventor is an open-source, free programming tool that is maintained by MIT to enable beginners to create software applications for Android and IOS. Just like Scratch and Tynker, App Inventor is a visual block-based programming language. It has two major windows which are the "component designer" and a "blocks editor." The component designer enables users to interface with the application they are developing, whilst the blocks editor is used to define the behaviors of application. Wolber (2011) stated that App Inventor adds real-life impact to developing a program, because it provides the ability to create applications based on users' real-life needs. App Inventor also provides opportunities to those

who are feeling creative and eager to create solutions to real-life problems through the use of mobile technology applications (Pokress & Veiga, 2013). Moreover, App Inventor is deemed useful because it creates an environment in which students can open their minds to consider their future career direction as they become mobile application problem solvers.

Alice

Developed by Carnegie Mellon University, Alice is a block and microworld programming language that engages learners using 3D graphics, enabling them to develop interactive stories, animations, as well as animated videos and games. Its purpose is to enable students from elementary education right through to university level to create computer animations using 3D models. Named after Lewis Carroll's classic, "Alice's Adventures in Wonderland," the application was designed to be easy to use and a fun coding experience, with both learners and instructors having benefit from Alice. With its wide-ranging application, Alice has been applied at the introductory level right through to actual programming, and especially within the scenario of "storytelling." With its IDE association, there is no need for users to remember syntax, which is a considerable advantage for both younger learners and beginners of any age. Reports have stated that Alice has been used not only at the K-12 level, but also for undergraduate studies in order to provide learners with the opportunity to succeed in the basic "Computer Science 1" course (Fincher et al., 2010). Alice is a free programming tool, and is supported by Oracle, Disney, Google, Microsoft, Intel, Pixar, Python, and the National Science Foundation.

2.3 Interdisciplinary Approach in Education

Interdisciplinary teaching/learning refers to the collaborative efforts involving multiple disciplines in order to understand a problem (Cassel, 2011; Mahadev & Conner, 2014). According to Repko and Szostak (2020), "*interdisciplinary studies is not a passing fad, it is here to stay*" (p. 23). Interdisciplinary instruction makes learners force their cognitive abilities and facilitates higher-order thinking skills as

they have to consider multiple disciplines. Due to its significant benefits, curricula based on the IA is growing rapidly. Research has shown that interdisciplinary studies provide learners with the chance to gain abilities in the awareness of research or researcher bias, thinking critically about topics, tolerating uncertainty due to collaborative team-studies and working arrangements, and increased mindfulness of research ethics (Borrego & Newswander, 2010; Cassel, 2011; Newell, 1992). Learners may recall prior knowledge easier because the IA encourages them to think in more than one discipline.

Establishing a bridge between two disciplines while teaching can be a very effective method of enhancing the learning process as it provides students with a diverse viewpoint on various topics during their learning activities. Interdisciplinary projects require students to think broader, helping them to approach problems with creative solutions that involve multiple disciplines. Goldsmith et al. (2018) studied the importance of the IA in education, and indicated that having a singular discipline in lessons limits the perspectives of learners, because they start thinking only in one particular norm without questioning any connections to other disciplines. However, the involvement of multiple disciplines in education can activate and foster deep understanding of issues and topics, because it challenges learners to demonstrate significant effort in order to correctly establish connections. Since interdisciplinarity helps to promote the understanding of complexity (Newell, 2007), it can establish an environment whereby the curriculum facilitates learning through the collaboration of multiple disciplines.

Lou et al. (2011) stated that real-world problems can be solved better with multidisciplinary thinking methods, and that STEM education as established in order to cope with this level of complexity by integrating multiple disciplines. This approach can provide students with the opportunity of applying knowledge from different areas into practice. Multidisciplinary collaboration is regularly applied in the solving of real-world problems, and its effect on education was established during the late 20th century (Nicolescu, 1999). Today's educational perspective requires

individuals to build knowledge through questioning, researching, and analyzing varied sources of information found via their educators. With the integration of the IA in education, information from different disciplines is combined in order to make the new knowledge more meaningful to the learner (Aydın & Balım, 2005).

The most important part of integrating the multidisciplinary approach into curricula is building upon new information by giving different meanings to knowledge sourced from other disciplines. As such, students may not feel themselves as applying a certain method, but that they will improve their own means of learning in order to obtain knowledge (Borrego & Newswander, 2010; Goldsmith et al., 2018; Newell, 1992). Therefore, the interdisciplinary teaching approach can be used in all different levels of educational curricula as a key method that develops knowledge from a lower to a higher level (Jones, 2010), with interdisciplinary methods helping learners make knowledge permanent and aid their goals in lifelong learning (Cai & Sankaran, 2015).

2.3.1 Interdisciplinary Collaboration of Mathematics and Computer Science

Interdisciplinary collaboration while teaching programming is considered to be a very effective means of reducing the intensity of learning a technical subject. Burns et al. (2012) stated that interdisciplinary collaboration in programming learning is shrouded in controversy, with instructors aiming to outline the importance of having a second discipline to find out how the mode of transferring knowledge to learners rendered as different from traditional teaching. Since programming is a complex computer science discipline where learners are taught how to solve complex problems by designing and creating computer programs (Celedón-Pattichis et al., 2013), the discipline is actually applicable to all activities that entail the giving of instructions to various computer aspects, hence the creation of computer programs plays a crucial role in technology (Guzdial, 2015).

Mathematics, on the other hand, is a broad discipline in which individuals learn to use critical thinking and spatial reasoning in order to solve mathematical problems (Park & Mills, 2014). In K-12 education, mathematics and computer science are addressed as two distinct subjects; although in actuality, the two disciplines have more in common than many would assume, with modern computer science topics dealt with using techniques that are related or derived from mathematics. The most common attribute of programming is the definition of logic statements to reduce the complexity of solutions to various real-life true or false statements. The mathematical representation of this programming aspect is the use of numbers "1" and "0," where "1 implies true" and "0 implies false". Cheng (1972) indicated that both mathematics and computer science somehow interacts with other disciplines; however, the relationship between these two sciences is undeniable and much more than just systematic.

First, the basic level where mathematics manifests in computer science is in the binary system of communication. Furthermore, emphasizing this relationship in K-12 education is paramount. Essentially, computers communicate mostly with their constituent parts and other computers using binary algorithms. This system of communication is made up of a sequence of 0s and 1s, which computers create and decrypt as a means to sending and receiving messages (Amiri & Nikoukar, 2011). Thus, in order to acquire a basic understanding of the workings of a computer, a simple understanding of mathematics is needed. In short, emphasizing this connection at the K-12 level is important as a means to encouraging students to embrace both disciplines. Herbsleb (2005) stated that in order to cope with the complexity of programming, disciplines related with the features of programming, such as mathematics, should be included as a means to understanding the content and to increase and make easier the delivery of knowledge. Scholars have found that individuals with an understanding of mathematics are noted to achieve a deeper understanding of programming, and the literature shows that a connection between mathematics and computer science facilitates learning, because both disciplines are

aimed at understanding a problem and build their solutions logically (Burns et al., 2012; Graham & Fennell, 2001; Lu & Fletcher, 2009; Pruski & Friedman, 2014).

Second, the overall link between computer science and mathematics is emphasized according to the role of the computer as a "calculator" and problem solver. However, K-12 students need to learn that computers alone cannot solve problems, and that they require the application of conscious knowledge. Computers were created in order to make certain tasks in life easier; for example, simplifying complex and lengthy calculations, identifying trends, and processing data On the one hand, these processes are based on largely mathematical data; however, computers can complete these tasks or solve problems through the application of mathematical calculations. These processes are conducted by computers at the logical level, hence, conscious intelligence or knowledge is needed in order to guide or guarantee the accuracy of the output data (Denning et al., 2017). It is for this reason that linking computer science and mathematics is deemed significant at an early stage in the K-12 process so as to prepare students who can use and interact with computers at both the logical and conscious levels.

Since mathematics is a discipline that entails comprehensive problem-solving interactions, this attribute creates an environment where individuals can learn more about problem solving, whilst dealing with various problem types. This necessitates a significant relationship between mathematics and programming, because programming entails the creation of code that attempts to solve various problems using computer applications (Burns et al., 2012). Knuth (1974) stated that many individuals working in computer science-related careers are also working in mathematics, because the core concepts behind a computer's algorithms are all about mathematics. Instructors with vast mathematical knowledge provide programming learners with problem-solving techniques that incorporate mathematical viewpoints, which is crucial in defining appropriate solution options. Problems related to mathematics provide learners with opportunities to test their programming skills according to real-life problems. This is attributed to the unprecedented rate of

technological growth as learners are becoming more innovative due to their exposure to creating solutions to problems that indirectly affect the society in which they live. Collaboration between programming and mathematics aids the creation of environments whereby learners are guided to be more innovative. The process requires facilitation to guide the application of their programming skills and technical interpretation of problems using mathematical viewpoints.

Scholars agree that a core of mathematics plays a significant role on the understanding of computer science, because both disciplines use powerful reasoning and find solutions through critical definition (Bruce et al., 2003; Havill & Ludwig, 2007; LeBlanc & Leibowitz, 2006). Mathematics entails understanding various problems and using powerful reasoning in order to develop and define mathematical solutions. As a joint department, the collaboration of mathematics and computer science increases the skills of learners' mathematical problem solving and programming in two ways, and is considered a means to broadening a learner's viewpoint. Put simply, interdisciplinary collaboration enhances the cognitive abilities of learners by empowering them with the ability to comprehend different viewpoints for a single scenario (Stozhko et al., 2015). Most programmers are inclined to a monotonous environment that comprises of long hours writing code (Jaccheri & Sindre, 2007). Mathematicians, on the other hand, use critical thinking to solve mathematical problems. Establishing a comparison between the two disciplines is a very effective way of merging the two personalities in order to create an individual who acquires a diverse viewpoint of the environment, the problems as well as the opportunities created by the problems, and their numerous programmatic and mathematical solution options.

In summary, there are several characteristics that link computer science and mathematics. The first is the binary nature of computer-based communication that uses mathematical concepts. The second reason is the purpose and function of the computer. Computers depend on mathematical input data in order to generate processed output that relies upon mathematical data and methods. Introducing

students early to these relationships at the K-12 level will give rise to a generation that thrive on big data and the technological era of the future.

2.3.2 Integrating Computational Thinking in Mathematics Education

Computational thinking can be related with many different forms due to having such a varied components list. Gadanidis et al. (2017) stated three main areas where computational thinking can be used, as computer programming through visuals on a screen, controlling digital forms such as robots and circuits, and in using algorithms during problem solving. Since computational thinking has a component of problem solving, it has a natural interaction with mathematics.

Educational leaders have sought to change the traditional system of mathematics education and to incorporate computational thinking skills, as opposed to the traditional procedural learning method in order to help today's students to be better equipped for their future career. Computational thinking has a clear role in this approach as it helps students to gain skills such as logical thinking, creating algorithms, and decomposition where students are able to break mathematical problems down into component parts in order to solve them (Berry & Csizmadia, 2016). Students can thereby solve mathematical problems from a different approach by creating algorithms through coding which help them to solve mathematical problems (Gadanidis et al., 2017; Martinelli, 2020). While creating these algorithms, students may discover new concepts that they may not have otherwise encountered in the procedural method of learning, thus helping them to develop critical-thinking and pattern-finding skills that can help elevate their level of understanding.

Integrating computational thinking into mathematic courses provides students with the opportunity to practice four main components; data practicing, modeling and simulating, creative problem solving, and systematic thinking (Weintrop et al., 2016). Therefore, the integration of computational thinking into mathematics education helps students to practice patience and persistence as they may encounter errors and challenges in their work due to enabling more than one skill at the same

time. Also, computational thinking integration provides the opportunity for more guidance from empirical research to both educators and students through STEM practices. Computational thinking provides students with a sense of achievement as they feel they have accomplished a significant task, having solved mathematical problems from scratch and having gained a much deeper understanding of the topic at hand. It also helps them to acquire skills that may prove of significant use in the future while job seeking in a world that is continually evolving and with everything shifting towards a more digitalized era.

Wing (2006) highlighted that the integration of computational thinking is important so as to facilitate the skills of reading, writing, and arithmetic, which ultimately drive a child's analytical ability. Mathematics is considered to play an important role in analytical ability because it acts as a bridge between critical thinking and problem solving. Supporting students' mathematics education at the elementary level of schooling, Gadanidis (2017) suggested the integration of computational thinking in mathematics courses because it supports students not only in their problem-solving skills, but also in their abstraction and automation skills as well. This integration also helps to create a bridge between the students and their teacher; thus, creating an avenue whereby the teacher can fully understand and better see how their students are thinking in response to a given problem (Gadanidis et al., 2017; Lee et al., 2020; Weintrop et al., 2016). This helps teachers to provide a personalized level of help, with every student taken care of and their level of understanding significantly increased as a result. Integrating these two disciplines also helps to ensure that students understand the errors they may encounter and why certain problems are solved in a certain way. Finally, it supports the teacher's ability to teach by providing various options and avenues through which they can fully explain certain concepts to their students. Furthermore, teachers should also be coached on computational thinking prior to implementation in order to ensure its success because their knowledge on the subject may only be theoretical, and they may therefore be ill-equipped to adequately teach their students, which may add to the students' state of confusion over some aspects of their learning (Gadanidis et al., 2017). Teachers

should therefore aim to provide computational thinking as an integrated element rather than to attempt to teach it separately in order to help improve their students' understanding.

Teaching mathematics using the procedural method can at times appear tiresome, with some students failing to fully understand or grasp certain concepts, leading in the end to lower than anticipated academic grades. However, through the integration of computational thinking, coding has made the learning process that much easier; providing students with greater understanding of what such concepts are all about. The approach also helps ease the teachers' workload and provides further options where they better understand how to reach individual students and thereby to address issues affecting them on a personal level. In what is now pretty much a fully digitalized era, this step may prove instrumental in setting up today's students for life when it comes to computer skills.

2.4 Summary of Literature

The focus of the current study rested on three main topics, that of computational thinking, computer programming education at an early age, and interdisciplinary collaboration in teaching.

Computational thinking is defined as a process which activates problem solving by involving critical thinking, algorithmic thinking, and also real-life activities (Berry, 2015; Grover et al., 2014; Hemmendinger, 2010; Wing, 2008; Yadav et al., 2014). Computational thinking skills are gathered under six main components, which are problem reformulation, recursion, problem decomposition, system testing, abstraction, and algorithm design (Shute et al., 2017; Wing, 2006). Thus, computational thinking is an ability of thinking, where an individual has to employ their creativity by including these six components in order to solve a given problem. In addition, computational thinking becomes not just the analyzing and solving of a given problem, but facilitates the learning of new knowledge due to its inclusion of real-life experience in the problem solving process.

The second topic of focus in the current study is computer programming education at an early age. Since technology usage has rapidly expanded on a global scale, and particularly in response to the COVID-19 pandemic, having knowledge of programming has become seen as a distinct advantage. However, both the teaching and learning of computer programming has been said to be difficult, due largely to the requirements of the skills involved (Bennedsen & Caspersen, 2007; Bornat & Dehnadi, 2008; Guzdial & Forte, 2005; Jenkins, 2002; Mazman & Altun, 2013; Özmen & Altun, 2014). Thus, teachers have sought new ways of teaching programming that are easier, not just for younger students, but also at the undergraduate level too. Guzdial (2015) stated that learning programming at an early age is important due to its relation to many new employment areas, productivity, broadening participation, learning about the world, computational thinking, and also computational literacy. With concern to the learning of programming being related to computational thinking, new techniques of teaching programming need to be developed so as to facilitate such learning and to develop learners' computational thinking skills at the same time. This raises the question of how to teach young learners programming. Based on this premise, researchers have focused their studies on, among other things, how to improve different skills' fields through the integration of interdisciplinary collaboration. According to Çınar and Tüzün (2017), disciplines that involve robotics, programming-based building blocks, and digital gaming can activate learners' computational thinking skills.

Interdisciplinary collaboration in education is the third main focus of the current study. Teaching and learning through an interdisciplinary approach refers to collaborative efforts involving multiple disciplines in order to understand a given problem (Cassel, 2011; Mahadev & Conner, 2014). Establishing a bridge between two or more disciplines while teaching can be a very effective method of enhancing the learning process as it provides students with a diverse viewpoint on various topics during their learning activities. The most important part of integrating the multidisciplinary approach into current curricula lies in building upon new information by providing different meanings to knowledge sourced from other

disciplines. As such, students may not feel themselves as applying a certain method, but will improve their own means of learning in order to obtain knowledge (Borrego & Newswander, 2010; Goldsmith et al., 2018; Newell, 1992).

In the current study, mathematics was chosen as the second discipline, since it shares a certain commonality with programming and also with computational thinking. When the literature is reviewed, it can be seen that a core of mathematics plays a significant role in the understanding of computer science, and that this is because both disciplines use powerful reasoning in aiming to find viable solutions through critical definition (Bruce et al., 2003; Havill & Ludwig, 2007; LeBlanc & Leibowitz, 2006). Therefore, establishing collaboration between mathematics and programming provides an opportunity for learners to develop both their both programming and computational thinking skills.

CHAPTER 3

METHODOLOGY

This chapter presents the design of the study and explains the procedures followed throughout the study. In detail, the chapter explains the research questions that were formulated, the design of the pilot study, lessons, and the experimental research, as well as informing about the participants of the study, the data collection methods and procedures employed, and how the study's data was collected and analyzed.

3.1 Research Questions

The current study is based on a primary research question, with four qualitative plus two quantitative sub-questions. These primary and sub-research questions are as follows:

1. What are the effects of teaching programming with a traditional approach (TA) and an interdisciplinary approach (IA) on secondary school students' attitudes, computational thinking skills, programming skills, and academic achievement level?

    a. Is there a significant difference between the TA and IA groups according to their Mathematics Attitude Scale scores (MAS)?

    b. Is there a significant difference between the TA and IA groups according to their Computational Thinking Skills Self-Efficacy Scale (CTSSES) scores?

    c. Is there a significant difference between the TA and IA groups according to their Computer Programming Self-Efficacy Scale (CPSES) scores?

d. Is there a significant difference between the TA and IA groups in their retention of Small Basic Programming Achievement Test (SBPAT) results?

e. What do teachers think about the teaching of visual programming tool using TA and IA methods?

f. What do students think about visual programming tool and its processes?

3.2 Participants

A secondary school in private schools is chosen for the study and the participants of the study were eight seventh-grade classes from the METU Development Foundation Secondary School (ODTÜ Geliştirme Vakfı Özel Ortaokulu). Cluster random sampling is used as sample method because the groups were already exist according to their level. The age of participant students was from 12 to 13 years old. The academic characteristics of each class were considered to be similar, as the school operated a policy of mixing classes homogeneously each year according to the students' GPA results in order for classes to be academically balanced. As researcher didn't have taken any role during the teaching process because of the researcher bias, two ICT teachers from the school took part in the study, and each teacher was responsible for four seventh-grade classes. In order to achieve a fair and balanced result, both of the teachers taught two classes using traditional teaching methods and two classes using the IA of the experimental process. Therefore, both ICT teachers taught two experimental and two control groups which were assigned randomly.

The sample of the study consisted of 192 students, split as 95 female and 97 male. However, due to the lack permission granted from the family of four students, their data were excluded from the study. Therefore, the study's analysis was conducted with an equal spilt of 94 female and 94 male students, totaling 188 participants. Since the sample size is large as it is 188, the results can be valid for all 7^{th} grade students

in Turkey. The summary of the participant students are provided in Table 2, Table 3 and Table 4.

Table 2. Demographic Summary of the Students In Terms of Groups

	Group	***n***
Experimental group	E1	24
	E2	24
	E3	24
	E4	22
Control group	C1	24
	C2	23
	C3	24
	C4	23
Total		188

Table 3.Gender

	n	***%***
Female	94	50
Male	94	50
Total	188	100

Table 4. Mathematics GPA Level

	n	***%***
Level 1	110	58.5
Level 2	56	29.8
Level 3	22	11.7
Total	188	100

According to Table 3 in terms of both genders were equal. Also, as can be seen, half of the participants formed the experimental groups whilst the other half formed the control group.

The mathematics GPA levels are grouped in three level according to Turkish grading system. Level 1 represents high level students' grades from 80 to 100, level 2 represents middle level students with the grade from 79 to 55 and Level 3 represents low level mathematic students with the grades from 54 to zero. When the students' academic achievement levels in mathematics are examined, it can be seen that 22 students had a "Level 3" which is low level of academic achievement, 56 had a "Level 3" which is middle level, whilst 110 of the participant students had a "Level 1" which is high level of academic achievement.

The students' average mathematics achievement was calculated as $M = 84.31$ ($SD = 10.48$), and each class in their respective groups had an average of between 80 and 100. Therefore, the similarity between the groups in terms of the students' mathematics GPA score was almost the same. Based on the 100-scoring system, the lowest achievement average among students was 43, while the highest was 99.

The students' background in programming and programming languages of each group was also close to each other. The data taken from the interviews is presented in Table 5.

Table 5. Students' Pre-Course Programming Language and Tools

JavaScript	Kodu Game Lab	Scratch
O-Bot	Arduino	Mindstorm
mBlock	C++	iDEA
C#	Visual Basic	Python

3.3 Design of the Study

The methods most regularly used in social science studies are qualitative, quantitative, and mixed method. Qualitative research is used to show the power of a research study by way of statistical procedures (Strauss & Corbin, 1990), and aims

to answer questions about “why,” “how,” or “what” concerning the collected data. Quantitative research, on the other hand, employs the measurement of objectives based on the application of surveys and/or questionnaires to test various hypotheses through either descriptive or inferential statistics (Denzin & Lincoln, 2011; Loewen & Plonsky, 2015; Taguchi, 2018). Quantitative research therefore focuses on finding answers to questions of “how many” or “how much” based on numerically larger pools of collected data. Studies conducted according to qualitative research methods employ a narrow focus using data from small samples to specific persons, whilst quantitative research involves larger volume data where the statistics are sourced from whole information drawn from statistically larger representative samples rather than the perception of individuals (Curry et al., 2009; Venkatesh et al., 2013).

The strengths of quantitative research are indicated as; the capacity to identify relationships between variables, a method of testing hypotheses, and the power to conduct group-based comparison (Castro et al., 2010; Strauss & Corbin, 1990; Venkatesh et al., 2013). On the other hand, qualitative research’s popularity is based on its strengths for associating data-conceived results to real-world context according to variables such as participants’ gender, beliefs, behaviors, and feelings (Curry et al., 2009; Venkatesh et al., 2013). Also, the potential for generalization based on the consistency and reliability of the collected data are other perceived strengths of the quantitative approach through the provision of detailed information from multiple methods on one or more subjects requiring careful examination, but with the added advantage of cost efficiency.

Besides the various strengths of both the qualitative and quantitative research methods each have inherent weaknesses too. The qualitative methods of study are considered difficult for the purposes of generalizing the result for the population of the community being investigated (Bektik, 2017; Castro et al., 2010). Moreover, since standard categories do not necessarily fit all formats, it is not always easy to analyze qualitative data, and the collection of such data requires more time than for quantitative data collection. On the other hand, quantitative data has certain

limitations too, such as it not always being easy to interpret the data as sometimes the results may be too complex to readily explain the content, and thereby the research may become too time consuming in terms of both its data collection as well the analysis (Taguchi, 2018).

Therefore, as both quantitative and qualitative research methods have their respective strengths and weaknesses, the mixed-method research approach can be applied in order to bridge both methods (Leech & Onwuegbuzie, 2009). Tashakkori and Teddlie (1998) explained the mixed-method approach as a form of research which focuses on the whole study by employing both qualitative and quantitative methods, either working independently or sequentially. For instance, a study may involve interviews as a form of qualitative data collection and also a survey that serves as a means for quantitative data collection. Creswell and Clark (2007) pointed out four major design types for the application of mixed-methods research. The first type is "triangulation," which is the most well-known approach, and involves both quantitative and qualitative data collection to form an overall deep understanding of the problem. The second major type is the "embedded" design, using either qualitative or quantitative data to interpret the problem. The third major design is "explanatory," which involves collecting data through the qualitative method which are then further explained through the quantitative method. The fourth major type is "exploratory," which is the opposite of the explanatory method as it collects data quantitatively and then explains the results through a qualitative approach. Scholars have explained the value of the mixed-method research design as a combination of two important methods that has the ability to highlight research results both as explanatory and confirmatory from the same study (Curry et al., 2009; Leech & Onwuegbuzie, 2009; Teddlie & Tashakkori, 2003; Venkatesh et al., 2013).

In the current study, Creswell's (2013) "sequential descriptive design" was chosen. In the sequential descriptive design, quantitative data is given priority, and is used in order to increase the structural role of the qualitative data.

The current study was conducted with a design using eight groups; with four control groups and four experimental groups.

3.3.1 Experimental Research Application

The experimental design is a study that is based on the application of experimentation principles and techniques in order to evaluate cause-and-effect relationships. Experimental research is developed on the premise of testing hypotheses under a controlled environment. Experimental research designs have two main categories, "true experimental design" and "quasi-experimental design." In a true experimental research design, statistical analysis is used as the primary scientific tool for the purposes of rejection or confirmation of the formulated research hypotheses (Bhatt, 2010). On the other hand, the quasi-experimental research design is similar to the true experimental methodology, in that it also employs experimental and control participant groups during the study. However, the quasi-experimental methodology does not employ random distribution in the assignment of various sample members into the experimental and control groups.

For the current study, the quasi-experimental design was selected to be used because the assignments were not distributed randomly, since the Small Basic programming language was taught both to the control groups, using the traditional approach subjects' method, whilst the experimental groups were taught using the interdisciplinary approach.

3.4 Experimental Processes

The experimental process was conducted to the study. Researcher worked with 8 groups and four of them are chosen as experimental while the other four group is chosen as control. The study process was designed separately for both groups. Both groups were taught with the same lesson plan at the first two weeks of the study because it was covering the user interface of the Small Basic and the basic

programming commands. For the following weeks, groups were taught with different lessons plans as experimental group were taught programming with the integration of mathematics as interdisciplinary approach (IA) while control group was taught with traditional method (TA).

The dependent variables of the current research study are the participant students' mathematics attitude, their computational thinking skills self-efficacy, as well as their programming skills and basic programming achievement. In terms of the independent variable, for the current study it is the interdisciplinary instructional approach based on mathematics.

The current study aimed to investigate the effect of teaching programming skills through an IA in collaboration with a course in mathematics. The teaching process combined activities from a Mathematics course with a Computer and Instructional Technology course in order to observe the aforementioned effect. The IA process was employed in the teaching process of the experimental groups, whilst the TA (subject-specific lecturing) was applied with the control groups.

In the study, both the experimental and control groups were assigned randomly. While the students in the experimental group were gathered into four distinct groups, similarly, four distinct control groups were formed. Measurements related to the dependent variables were performed prior to and following the experimental process application. During the experimental process, two ICT teachers worked alongside the researcher, each teacher had classes as two control groups and two experimental groups.

3.4.1 Pilot Study

A pilot study is a small-scale study applied in order to test a main study. Specifically, a pilot study tests the research protocols of the main study in order to detect and problems, and to improve them according to feedback and/or data results prior to the application of the main study. It is considered as a preparatory action for the main

study so as to enable the researcher to be made aware of any problems in the research methodology prior to its actual application. The literature highlights that conducting a pilot study prior to a main study is an important stage, as it is of significant benefit to identify and improve upon any potential problem areas so as to strengthen the overall result of the study in hand (Hassan et al., 2006; Kraemer et al., 2006; Lancaster et al., 2004).

The pilot study is seen as essential because it controls the feasibility of the main study, and helps to make the research questions clearer. It also helps to focus the process of the main study because researchers can see in advance any problems associated with the management, communication, application, and data collection techniques employed during the pilot study. Therefore, the advantages of a pilot study are highlighted as purification of the instruments to be employed, the opportunity to open the researcher's mind to alternative techniques related to the main study based on a real (small-scale) application, to help determine the feasibility of the study which may save time and increase efficiency during the main study, and to improve on the planned data collection process, instruments, and procedures if needed (Lancaster et al., 2004; Merriam & Tisdell, 2015; Yin, 2009).

In the current study, a pilot experimental study was applied at the same school 1 year prior to the execution of the main study. The reasoning for the long break between the pilot and main study was that the topic of study was only taught during a single time period each academic year. The pilot study lasted for 8 weeks in total, during March and April of 2017. The pilot study was applied to six different seventh-grade classes, with each lesson lasting 40 minutes (1 classroom hour). Since the topic was compulsory in the seventh-grade national school curriculum, all students in the classes participated in the study. According to school's policy, each class was designed and composed homogeneously in terms of the students' annual GPA, as a means to ensuring a balanced academic student profile (based on GPA) within each class. Therefore, the classes included in the pilot study were each already homogeneously distributed by the school rather than the researcher.

In the execution of the current study's pilot study, six different lesson plans were applied each week between the experimental and control groups. Lesson delivery were by way of direct instruction, with students expected to finish each given task. However, the pilot study showed that verbal description of the assigned problems was inadequate, therefore, PowerPoint presentations were designed for each lesson plan in order to describe the problems with more clarity to the students through the use of illustrations. Also, during the pilot study, no interviews were held with the participant students. However, following a meeting held with the participant ICT teachers and the researcher's thesis advisory committee, the need for student feedback was raised, and student interviews subsequently planned for inclusion in the main study.

3.4.2 Main Study

The main study was conducted between February and June of 2019 at the ODTÜ Geliştirme Vakfı Özel Ortaokulu (METU Development Foundation Schools), Ankara, Turkey. The main study lasted for a total of 10 weeks, plus 1 week to conduct an achievement test 2 months following the application in order to measure the participant students' learning retention. Prior to the main study, permission was received to conduct the study from the school, the parents of the students who participated in the study, the Ethics Committee of Middle East Technical University, and the Turkish Ministry of National Education. Two ICT teachers employed at the school taught the participant groups in a combined total of eight classes, divided as four classes for both the experimental and control groups. Each of the two ICT teachers had four seventh-grade classes, so that each taught lessons according to the TA (for two control groups) and also the IA (for two experimental groups). Since the topic of study forms a compulsory element of the seventh-grade national school curriculum, the 10-week study period was all applied in a single phase. An outline of the activity plans prepared by the researcher for both the experimental and control groups. A sample from activity plan of week 6 is presented in Table 6. respectively.

The full plan is added on appendices.

Table 6. Activity Plan for the Groups

Week	Topic	Acquition (Control Group)	Acquition (Experimental Group)
6	**Lesson Plan 5:** For Loops	**Programming achievements:** • Remembering the reasons for using loops. • Remembering program lines are shortened with loops. • Understanding the difference between "while" and "for loop."	**Programming achievements:** • Remembering the reasons for using loops. • Remembering program lines are shortened with loops. • Understanding the difference between "while" and "for loop." **Mathematics achievements:** • Expressing rule of number patterns by letter. Finding the questioned term of a pattern whose rule is expressed by a letter.

During Week 1, the participants were applied pretests using three instruments; the Computer Programming Self-Efficacy Scale, the Mathematics Skills Self-Efficacy Scale, and the Computational Thinking Skills Self-Efficacy Scale. In Week 2 and Week 3, both groups were taught the same lesson plan as it concerned how to use the actual programming tool and its user interface. From Week 4 to Week 7, the groups were taught according to different lesson plans, with each receiving different instruction and examples according to prescribed teaching approach. In Week 8, posttests were applied to both groups, and in Week 9 the students were applied an achievement test. In the final week of the process, Week 10, interviews were conducted with both the participant students and their teachers. Whilst the main study concluded at the beginning of April, during the first week of June, a further achievement test was applied in order to measure and evaluate the students' learning retention.

3.5 Data Collection

Data is arguably the most important core process of the study, and is used to examine the value and validity of the study. A method of data collection refers to the use of techniques applied for the collection of a study's research data (Harrell & Bradley, 2009). The purpose of the current study required both qualitative and quantitative data to be collected, hence a mixed-method research approach was adopted. Data were collected through self-efficacy scales, achievement test results, and the participant students' mathematics GPA scores as the quantitative data, and via interviews conducted for the qualitative data collection.

3.5.1 Instruments

This subsection explains the process of how each of the five distinct data collection tools were used. Four of the tools were instruments, plus two variations (student/teacher) of an interview protocol form. Three of the tools (Mathematics Attitude Scale, Computational Thinking Skills Self-Efficacy Scale, and the Computer Programming Self-Efficacy Scale) were developed by other researchers. The fourth was the BASIC Programming Achievement Test that was developed for the current study by the researcher. In addition, two interview protocol forms were developed by the researcher in order to support the collection of the study's qualitative data element.

3.5.1.1. Computer Programming Self-Efficacy Scale (CPSES)

In order to examine improvements in the participant students' programming skills, a Computer Programming Self-Efficacy Scale was applied both as a pretest and again as a posttest. The scale was originally developed by Wiedenbeck and Ramalingam (1999), and was adapted to the Turkish context by Altun and Mazman (2012). In the original scale, Wiedenbeck and Ramalingam (1999) had 32 items in their scale, designed as a 7-point, Likert-type instrument with nine sections under a two-factor

arrangement ("ability to perform simple programming tasks" and "ability to perform complex programming tasks"). The Cronbach alpha coefficient of the original scale was found to be .928, with the scale applied as a pretest to a total of 421 students at the beginning of a C++ programming course. Whilst adapting the scale into Turkish, Altun and Mazman (2012) changed the language of the scale into a more general state in order that it could be applied to any programming language. Also, they decreased the number of items in the scale to nine according to their test results.

For the current study, Altun and Mazman's (2012) Turkish version of the Computer Programming Self-Efficacy Scale was employed, formed as a nine-item, 7-point, and Likert-type scale.

3.5.1.2. Mathematics Attitude Scale (MAS)

The Mathematics Attitude Scale, as developed by Duatepe and Çilesiz (1999), was used in the current study in order to determine the participant students' attitudes and confidence towards mathematics.

The scale was originally applied to 230 samples during a pilot study conducted by the developers. Whilst developing the scale, Duatepe and Çilesiz (1999) focused on the individuals' love, profession, fear, materiality, interest, pleasure, and trust towards mathematics, therefore, the scale was developed according to these seven dimensions.

During the development of the scale, the 44-item, seven-dimension draft attitude scale was applied to 230 mathematics, engineering, administration, and education students from the Middle East Technical University in Ankara, Turkey. According to the pilot study's analytical findings, six items were subsequently deleted from the scale. Duatepe and Çilesiz's (1999) analysis showed that the revised scale consisted of four dimensions. In total, there were 13 items in the first dimension to measure enjoyment of mathematics, nine items in the second dimension to measure confidence and anxiety, eight items in the third dimension to measure occupational

and daily importance of mathematics, plus eight items in the fourth dimension to measure interest, liking in mathematics. According to their analysis results, item-scale correlation proved the scale's validity. The smallest correlation value for the first dimension (.55), whilst for the second dimension it was .62, for the third dimension is was .48, and for the fourth dimension is was .51. As to the reliability analysis, the Cronbach Alpha coefficient was found to be .96 (Duatepe & Çilesiz, 1999).

In order to test the validity of the scale, Exploratory Factor Analysis (EFA) was performed. As a result of the analysis of the principal components rotated according to the principal axis, the scale's 38 items were grouped to form four dimensions. If a load of a scale item on a factor is above .48, the item is counted for that factor. When assigning an item on a scale to a factor, attention was paid to the difference between the load on that factor and the load on other factors above a certain value. Their findings showed that the factor loads were distributed between .48 and .80. The values regarding the items in the subscales and their factor loads are given in Table 7.

Table 7. Item/Factor Loads for MAS

Item	Factor 1 Enjoyment	Item	Factor 2: Confidence & Anxiety	Item	Factor 3: Occupational & Daily Importance	Item	Factor 4: Interest, Like
31	.71	20	.80	4	.76	6	.79
2	.70	26	.77	14	.76	16	.76
15	.69	21	.77	17	.71	19	.73
34	.67	35	.76	12	.71	25	.69
30	.66	26	.75	37	.69	30	.67
10	.65	18	.75	7	.63	11	.63
32	.64	1	.71	21	.61	3	.60
5	.64	33	.69	25	.48	23	.51
13	.64	27	.65				
9	.62						
28	.61						
38	.59						
8	.55						

The scale included 38 items, developed as a 5-point, Likert-type scale and was applied to the current study as both a pretest and as a posttest.

3.5.1.3. Computational Thinking Skills Self-Efficacy Scale (CTSSES)

The Computational Thinking Skills Self-Efficacy Scale was developed by Gülbahar et al. (2019) in order to determine students' computational thinking skills, and designed to be applied both as a pretest and then later as a posttest. The scale's authors applied the test to 952 secondary school students from different regions of Turkey. The results and analyses of their study formed a five-dimension scale, with a total of 39 items using a 5-point, Likert-type design (Gülbahar et al., 2019).

Gülbahar et al. (2019) found the KMO coefficient to be .966, and the Bartlett test significance value as < .05 level. The fact that the KMO value was above .50 and the Bartlett value was significant, meant that factor analysis could be performed on the dataset. As a result of their analyses, it is seen that the 39-item scale consisted of a five-factor structure (Algorithm Design Competency, Problem Solving Competency, Data Processing Competency, Basic Programming Competency, and Self Confidence Competency). From Confirmatory Factor Analysis (CFA) conducted on the model, three items were subsequently omitted from the scale and the final structure of the scale included 36 items. The corrected item-total correlations ranged between .632 and .386, whereas the Cronbach Alpha coefficients ranged between .762 and .930. Also, *t*-tests for the item average means of the bottom 27% and top 27% of the groups were presented as significant differences between those groups.

The values regarding the items in the subscales and their factor loads are given in Table 8.

Table 8. Item/Factor Loads of for CTSSES

Item	Factor 1: Algorithm design	Item	Factor 2: Problem solving	Item	Factor 3: Data processing	Item	Factor 4: Basic programming	Item	Factor 5: Self-confidence
S2	.823	S44	.733	S19	.802	S33	.644	S5	.636
S1	.816	S48	.710	S20	.762	S28	.601	S4	.615
S3	.811	S39	.671	S23	.595	S37	.586	S14	.614
S9	.795	S43	.654	S26	.565	S34	.571	S6	.553
S8	.760	S40	.651	S25	.550	S38	.501	S18	.547
S10	.729	S46	.631	S24	.546	S17	.497	S16	.497
S11	.722	S42	.623	S21	.539				
S12	.714	S47	.616						
S7	.674	S45	.569						
		S49	.505						
		S41	.504						

3.5.1.4. Small Basic Programming Achievement Test

The Achievement Test was adopted by the researcher for the current study. The questions that formed the Achievement Test were developed for the Microsoft Small Basic programming tool, and were adapted from those used in the study of Sáez-López et al. (2016), which was prepared for the Scratch programming language. Besides testing students' achievements, the test was developed by focusing on four dimensions of computational thinking skills, which are also common to the programming skills components.

The first dimension tests how students can set coding commands in the right order so as to demonstrate step-by-step algorithmic planning. The second dimension looks at debugging and the modification of incorrect code. The third dimension is completion, which involves students completing elements of incomplete code, whist the fourth dimension interprets what the students understand from some given code, and how they think it should be written on the screen. During the development of the Achievement Test, the researcher worked together with the two ICT teachers from

the METU Development Foundation Secondary School and applied changes deemed relevant according to their feedback.

3.5.1.5. Interview Protocol Forms

A qualitative research interview is defined as giving meaning to studies by describing connections between the real-world and the study in hand (Kivits, 2005; Kvale, 1983; Mann & Stewart, 2000; Opdenakker, 2006). Opdenakker (2006) indicated four different techniques for interviews, which are face-to-face, MSN messenger, telephone, and email. For the current study, the face-to-face interview technique was selected because of its advantage on social cues such as the opportunity for the researcher to observe the interviewees' body language, tone of voice intonation etc.

In the current study, two different interviews were conducted. The first was conducted with the study's participant students, whilst the second was conducted with the class teachers who applied the lesson plans to the study's participants.

Both the "Interview Protocol for Students" and the "Interview Protocol for Teachers" contained a total of seven questions. Similar steps were taken in the development of both interview protocol forms, with each developed jointly according to three stages.

In the first stage, the relevant literature was first examined in detail, and then interviews were conducted with field experts, resulting in a 12-item interview pool created based on their collective experiential feedback. For the second stage, items from the item pool were presented to four Measurement and Evaluation experts. A draft form was created based on the experts' feedback, and then submitted to the examination of linguists. Amendments were subsequently applied to the question roots in light of the feedback received from the experts. In the third stage, pilot interviews were conducted with one teacher and three students in order to test out the applicability and understandability of the interview question items. As a result of analyzing the data obtained from the pilot interviews, the final version interview protocol forms were completed. The final version Interview Protocol for Students

and the Interview Protocol for Teachers are both presented within Appendix D (in Turkish).

3.6 Data Analysis

The data analysis process started with the collection of the study's data. After receiving the appropriate permissions from the related individuals and institutions, the completed data forms were imported into Survey Monkey, an online survey tool used to collect quantitative data. IBM's Statistical Package for Social Science (SPSS, version 25) software was then used in order to analyze the collected quantitative data. The level of significance (*p*-value) was set as .05 by the researcher.

The qualitative data collected through the student and teacher interviews were analyzed using the Nvivo 12 software. First, by undertaking a descriptive analysis of the interview data, similarities between the students' views on coding teaching using the interdisciplinary (experimental) approach were revealed. Subsequently, codes were created according to the content analysis method. The codes were then grouped according to feature similarity under themes.

3.6.1 Trustworthiness

A study should be reviewed by peers or someone who has pertinent knowledge about the related topic in order to ensure the credibility of the research (Creswell & Miller, 2000). Trustworthiness was also highlighted as being an important criteria in qualitative research by Lincoln and Guba (1985), and encompasses four main aspects which are credibility, dependability, transferability, and confirmability (Lincoln & Guba, 1985). Credibility is described as linking the results of a study with that of the real world in order to show the trust and truth of the study by focusing on two main aspects; triangulation and member-checking (Lincoln & Guba, 1985). In order to clarify the trustworthiness of the current study, triangulation measures and member checking were employed.

3.6.1.1. Triangulation

Creswell (2012) defined triangulation as a qualitative research strategy used to show the value of a study in terms of its credibility and accuracy. The current study involved triangulation in different methods. First, during the data collection process, six different forms were used, including three scales, an achievement test, the participant students' GPA score, and a set interview protocol. Second, the interviews were conducted not just with the participant students, but also with the teachers who had delivered the lessons plans developed as part of the current study. In addition to the varied approach to data collection, intercoder agreement strategy was also applied in the analysis of the collected qualitative data.

3.6.1.2. Member-Checking

Member checking is defined as a way to follow increasing the creditability in qualitative research (Lincoln & Guba, 1985). To ensure whether the process followed fits with the theoretical instruction of ICT lessons in seventh grades, the lesson plans, tools and all the instruction were sent different ICT teachers. Birt et al. (2016) indicates that member checking is important in qualitative research because it is essential for researchers to be transparent about their study and member checking gives a transparency to the achievement, goals and methods of the study.

3.6.2 Quantitative Data Analysis

3.6.2.1. Reliability of the Scales

Two different internal consistency values were calculated based on the data obtained from the pretest and posttest applications of the three scales used in the current study. When the values presented in Table 8 are examined, the findings show that the three scales used in the study are reliable tools for the purposes of measuring students'

mathematics attitude, their computational thinking skills self-efficacy, and their computer programming self-efficacy skills.

The data obtained from the three data collection scales were analyzed using the SPSS program. Prior to making any further analysis, the assumption of normality of the data obtained from the experimental and control groups were tested. For this purpose, Kolmogorov-Smirnov and Shapiro-Wilk tests were performed and the resultant skewness and kurtosis values were examined. When the results are analyzed, it can be seen that in the normality distribution of the research data, the *p*-values are considered large ($p > .05$) in most of the subscales, and low ($p < .05$) in some of the subscales.

Whilst it is not considered a desired situation to have calculated values greater than .05, some studies conducted in the Social Sciences field have stated the opinion that if the desired values cannot be reached within the normal distribution of the data, then kurtosis and skewness values should be calculated in order to examine the normality assumption.

In the analysis conducted on this subject in the current study, it was observed that the expected kurtosis and skewness values for all of the scales and subscales were found to exceed the observed values. When results came from the scale is analyzed, it can clearly be seen that the average scores of the students in the experimental and control groups obtained from the scales showed normal distribution. Therefore, parametric test techniques were used in the data analysis of the research (Huck, 2008).

Covariance analysis and repeated measures variance analysis were conducted in order to reveal differences between the mathematics achievement of the participant students in the experimental and control groups and their mathematics attitudes, programming skills, and programming achievement tests (Akbulut, 2010; Hancock, 2004). For paired groups, paired samples *t*-tests, and for comparisons between groups, independent samples *t*-tests were used to compare the averages between the

experimental and control groups for each variable. Also, in order to see the effects on the study of each course teacher, who participated in the study as a class instructor, within group comparisons, between group comparisons, and covariance analyses were performed.

All of the data used in the study were analyzed according to a significance level of .05. In the normality test for the mean scores obtained from the scale items, it was observed that the data was normally distributed, hence, parametric tests were conducted.

3.6.3 Qualitative Data Analysis

LeCompte and Goetz (1982) stated that achieving complete validity and reliability for any research model is not possible. However, researchers can still approach the goal of making their research valid and reliable by balancing various factors that aim to increase the reliability and validity of specific known research problems and objectives. Using various strategies, researchers can reduce factors threatening a study's reliability and validity.

Scholars state that validity concerns the accuracy of the research results, whilst reliability relates to the reproducibility of the research results (LeCompte & Goetz, 1982; Miles & Huberman, 1994; Patton, 2014; Yıldırım & Şimşek, 2008). Both validity and reliability are based on two themes, external and internal. External validity focuses on seeing if the different groups obtain the same results by using a data collection tool, whilst internal validity focuses on the measurement, although the researcher is free to use this or not (LeCompte & Goetz, 1982). When it comes to reliability, external reliability checks to ascertain if the results of the research can be obtained in similar environments with the same processing, whilst internal reliability looks at whether other researchers could achieve the same results using the same data (LeCompte & Goetz, 1982).

3.6.3.1 Validity

For qualitative data analysis where verbal data is obtained to be processed, the concepts of validity and reliability are somewhat different than quantitative data analysis. According to Yıldırım and Şimşek (2008, p. 30), in qualitative studies; "proximity to the research area," "gathering detailed and in-depth information through face-to-face interviews," "gathering information directly and in the natural environment where the incident took place," "gathering long-term information," and "being able to go back to the field to confirming the findings and providing additional information" are all considered important features in terms of ensuring validity.

In order to ensure the internal validity in qualitative research, the following issues should each be considered carefully:

> As presented by LeCompte and Goetz (1982): *"Research findings should be meaningful considering the environment in which the data were obtained and defined depending on this environment," "Findings are consistent and meaningful in themselves," "The emerging concepts to forming a meaningful whole," "confirmation of findings using different data sources, data collection methods and analysis strategies,"* by Miles and Huberman (1994): *"The findings obtained according to different sources, methods and strategies form a meaningful whole," "the findings obtained are consistent with the conceptual framework or theory created," "proper use of some rules and strategies to confirm findings," "using alternative approaches to explain the findings,"* and by Yıldırım and Şimşek (2008) *"finding the findings realistic by the individuals participating in the research," "the estimations and generalizations made based on the findings are consistent with the data obtained."*

In the current study, in order to ensure the internal validity, the following steps were conducted:

1. Research findings were defined concerning and in relation to each other. In the reporting of the findings, direct quotations from the obtained research data were included;
2. In order for the findings to be consistent and meaningful within themselves and for the emerging concepts to form a meaningful whole, the data and themes collected under each theme had to constitute a meaningful whole among themselves;
3. In the analysis process, where data collection was followed by the researcher, the data were systematized, and the consistency of the data checked continuously;
4. Before and during the data collection process, the literature related to learning programming at an early age, students' attitude towards mathematics, computer programming self-efficacy, and computational thinking skills self-efficacy were examined continuously in order for the theoretical structure to guide the researcher in obtaining the findings;
5. Data obtained from the application were examined together with the theoretical structure, so that the theoretical framework guided the creation of the code list used in the data analysis;
6. Themes that emerged from analyzing the qualitative data were based on inductive analysis, with findings supported by the literature related to the comments. In presentation of the findings, some reported from quotes to results and comments, whilst others reported from results and comments to quotes;
7. While developing the interview protocol forms, the relevant literature was examined and expert opinion sought. Feedback regarding the interview

questions was sought from eight different ICT teachers as well as from assessment and valuation experts;

8. After the data were obtained, the researcher and the participants reexamined the data so that it was verified by the participants in order to reveal any misunderstandings;
9. Emphasis was reported that any generalizations made were based on the findings according to similar situations and conditions being applicable;
10. Stated generalizations made were limited to the data obtained from the participants and from situations experienced during the process. For certain generalizations, they were limited to findings or from the groups that participated in the practice.

External validity relates to the generalizability of a study's results. The process checks if the results are generalizable according to similar situations and conditions. Yıldırım and Şimşek (2008) stated that if results can be generalized, then the qualitative data is deemed to be externally validated. Even though it is important to validate a study externally, generalizability of qualitative research may still be considered as weak. Therefore, the results of a qualitative research study cannot be directly generalized to another situation. While generalization may be achieved directly in quantitative research, generalization can only be achieved indirectly in qualitative research. According to Yin (2009), generalization means analytical generalization. In other words, a qualitative researcher working with a limited number of participants or studying a certain situation may end up with a hypothesis, a conceptual model, or a new theory at the end of the research.

In the current study, the following steps were followed in order to ensure the external validity of the research:

1. The study group of the research and its characteristics, and details of the interview process were explained in detail in the methodology of the research;

2. Guided by the conceptual framework of the study, generalizations were reached according to the study's findings, the characteristics of the study group, and the environment. It was stated that generalizations were limited to within the framework of the specified features;
3. Details of the qualitative data process were carefully explained to the participants, with the processes described in detail in order that other researchers may benefit from the generalizations based on similar groups and environments;
4. The consistency of the theoretical framework, including the results and generalizations reached, was taken into consideration (LeCompte & Goetz, 1982; Miles & Huberman, 1994; Patton, 2014; Yıldırım & Şimşek, 2008).

3.6.3.2 Reliability

Reliability can result in certain problems in terms of sensitivity to the natural environment which can affect the importance of measured perceptions (Yıldırım & Şimşek, 2008). One of the basic principles of qualitative research is maintaining an awareness of the facts that constantly change depending on the each individuals involved and the environment. As such, researchers should accept from the outset of a study that repetition of the research in similar groups may not achieve the same results, because it is not possible to repeat an exact research due to human behavioral variables. Although certain criteria related to reliability contradict some qualitative research criteria, there are some precautions that can be taken in qualitative research, both for the purposes of ensuring a study's external reliability and also internal reliability.

In the current study, to ensure the internal reliability of the research, similar studies were examined from the literature. Accordingly, the following process steps were applied in terms of internal reliability:

1. Research questions were clearly stated, plus the method, process, data collection method, and the study's research instruments were explained in line with the study's research questions;
2. Research data were obtained by applying interview protocols developed according to the research questions. Data were first presented without adding comments, with related quotations from the data also presented. This approach tried to reflect the originality of the data;
3. Whether or not the research data answered the research questions and the consistency of the obtained results compared to the findings was continuously checked (LeCompte & Goetz, 1982; Miles & Huberman, 1994; Yıldırım & Şimşek, 2008).

After following these steps, the literature was reviewed prior to creating the draft code list. For the qualitative analysis, two intercoders analyzed the data obtained from the interviews in addition to the researcher. Following an explanation about the study given by the researcher, each intercoder coded the interview data separately, without contact with either the researcher or the other intercoder.

A draft code list was then discussed by the researcher and the two intercoders in order to create the final form. An example coding was created using the agreed code lists, and only after consensus was reached on the sample coding did the actual coding activity commence. The researcher identified common codes by comparing the coding made by each of the intercoders to the coding applied by the researcher.

3.7 Researcher Role and Bias

Polit and Beck (2014) described bias as an effect or effects which may contribute to failure seen in the results of a study. The researcher plays a critical role as the central instrument in an academic social sciences study (Creswell, 1994; Mehra, 2002; Yıldırım & Şimşek, 2008).

The researcher in the current study had prior knowledge and research experience on teaching as a professional, so possessed a good level of knowledge about what the needs of the target group's participants were. At the time of the study, the researcher had taught at the K-12 level for a total of 9 years, and had also studied the current research area academically since 2006. The researcher supported the concept of interdisciplinary collaboration, and believed that the involvement of a second discipline in the teaching of coding could increase students' programming skills instinctively. Due to the researcher's strong belief in the study, and in order to minimize any researcher bias, the researcher did not take an active role in the teaching of the participant students in the current study. The eight participant classes were taught by two different ICT teachers. Also, as another means to minimizing bias, the researcher worked with 10 different ICT teachers, two mathematics teachers, and four Measurement and Evaluation experts during the development of the lessons plans and instruments included in the current study. Therefore, it is suggested that the lesson plans of both the experimental and control groups were designed fairly. Additionally, the interview process and achievement tests were also reviewed by the experts in order to control any potential researcher bias in the current study.

3.8 Summary of Methodology

The current study was focused according to the research question, "What are the effects of teaching programming with a traditional approach (TA) and an interdisciplinary approach (IA) on secondary school students' attitudes, computational thinking skills, programming skills, and academic achievement level?" In order to address this question, six sub-questions were added to the study in order to find any significant differences between the TA and IA groups according to their Mathematics Attitude Scale scores, their Computational Thinking Skills Self-Efficacy Scale scores, their Computer Programming Self-Efficacy Scale scores, their retention of BASIC Programming Achievement Test results, and both the students'

and teachers' opinions about Small Basic Programming and its process. The sample of the study consisted of 188 students, split equally as 94 females and 94 males. With both qualitative and quantitative data, the mixed-method research approach was applied in the study, with a quasi-experimental design employed as the assignments were not distributed randomly.

In the study, both the experimental and control groups were assigned randomly, with four distinct experimental groups and four distinct control groups. During the preparation of the study and the lesson plans, 12 ICT teachers and two mathematics teachers took part. Also, while applying the experimental process, the researcher took on the role of mentor to the ICT teachers, and two of the ICT teachers worked alongside the researcher in teaching the students, with each assigned two control groups and two experimental groups. The study period lasted for a total of 10 weeks, plus 1 week used to conduct an achievement test 2 months following the application in order to measure the participant students' learning retention. Data were collected through self-efficacy scales, achievement tests, and the participant students' mathematics GPA scores as the quantitative data, and via interviews conducted by the researcher as the qualitative data of the study. The instruments applied during the study were the Mathematics Attitude Scale, the Computational Thinking Skills Self-Efficacy Scale, the Computer Programming Self-Efficacy Scale, the BASIC Programming Achievement Test, and two interview protocol forms.

CHAPTER 4

FINDINGS

The study was performed based on both qualitative and quantitative data in order to respond to the primary research question of the study: *"What are the effects of teaching programming with a traditional and an interdisciplinary approach on secondary school students' attitudes, computational thinking skills, programming skills, and academic achievement level?"* In this chapter, first the quantitative analysis results are described with detailed tables, illustrative graphics, and comments based on the scales applied and an achievement test. This is followed by the qualitative analysis results from the interviews conducted.

Semi-structured interviews were held with both the teachers who applied the study as prescribed to them by the researcher, and also with the students who participated in the study. The interviews not only aimed to reveal the strengths and weaknesses of the study, but also as a means to investigating the effect of the interdisciplinary programming teaching approach on the academic achievement of the experimental groups' students. In order to highlight any relation between the participants' and instructors' perceptions according to the study's research questions, the numerical data analysis results obtained from the quantitative scales and tests were interpreted according to the qualitative data collected from the interviews. Therefore, this chapter is divided into two sections, presenting first the Numerical Analysis Findings and then the Verbal Analysis Findings.

4.1 Quantitative Analysis Findings

Four of the six research sub-questions of the study are addressed in this section. The numerical data of the study were collected from the application of three scales, the

Mathematics Attitude Scale, the Computational Thinking Skills Self-Efficacy Scale, and the Computer Programming Self-Efficacy Scale, and also from the BASIC Programming Achievement Test.

The scales were applied as a pretest prior to the commencement of the study, and again at the conclusion of the study as a posttest. The BASIC Programming Achievement Test was also applied twice; once at the end of the study's application stage, and then a second time 2 months later so as to measure the students learning retention. All of the instruments were applied to both the experimental and control group students. The findings are presented in the following subsections according to the relevant research sub-questions.

4.1.1 Research Question A: Mathematics Attitude

This subsection presents the findings in response to the first research sub-question (a): *"Is there a significant difference between the control and experimental groups according to their Mathematics Attitude Scale scores?"*

The hypothesis for the first research questions is stated below:

> H_0: Integrating mathematics into programming courses increases student's mathematics attitude.
>
> There is no significant difference between the experimental and control groups to their mathematics attitude.
>
> H_a: Integrating mathematics into programming courses does not increase student's mathematics attitude.
>
> There is a significant difference between the experimental and control groups to their mathematics attitude.

The mathematics attitude of the experimental and control groups' students was examined in order to answer the first research sub-question. For the purpose of determining whether or not a significant difference exists between the mathematical attitude of the students from the first and second application, paired group *t*-tests were performed. This procedure was applied both for the experimental and the control groups. Beside the paired group *t*-test, independent samples *t*-tests were performed so as to compare the scores from the pretests and posttests of both the experimental and control groups. The results of the two Mathematics Attitude Scale applications are presented in Table 9.

Table 9. Paired Groups t-Test Differences Between Mean Values from MAS Scores

Pair	Test Dimension/Scale (Group)	*M*	*SD*	*t*	*df*	*p*
Pair 1	Pretest Interest, Compassion, Pleasure (Experimental)	2.87	0.46	1.785	93	.08
	Posttest Interest, Compassion, Pleasure (Experimental)	2.75	0.52			
Pair 2	Pretest Interest, Compassion, Pleasure (Control)	2.65	0.50	**-3.130**	**93**	**.00****
	Posttest Interest, Compassion, Pleasure (Control)	2.85	0.43			

Table 9. Paired Groups t-Test Differences Between Mean Values from Mathematics Attitude Scores (continued)

Pair	Test Dimension/Scale (Group)	*M*	*SD*	*t*	*df*	*p*
Pair 3	Pretest Fear, Trust (Experimental)	3.00	0.43	**2.996**	**93**	**.00****
	Posttest Fear, Trust (Experimental)	2.83	0.28			
Pair 4	Pretest Fear, Trust (Control)	2.69	0.41	-1.244	93	.217
	Posttest Fear, Trust (Control)	2.76	0.36			
Pair 5	Pretest Professional Importance (Experimental)	3.12	0.46	**4.886**	**93**	**.00****
	Posttest Professional Importance (Experimental)	2.87	0.20			
Pair 6	Pretest Professional Importance (Control)	3.06	0.43	**2.333**	**93**	**0.02***
	Posttest Professional Importance (Control)	2.93	0.29			
Pair 7	Pretest Pleasure (Experimental)	3.12	0.50	1.292	93	0.2
	Posttest Pleasure (Experimental)	3.03	0.49			
Pair 8	Pretest Pleasure (Control)	3.12	0.52	**5.345**	**93**	**.00****
	Posttest Pleasure (Control)	2.81	0.47			

Table 9. Paired Groups t-Test Differences Between Mean Values from Mathematics Attitude Scores (continued)

Pair	Test Dimension/Scale (Group)	*M*	*SD*	*t*	*df*	*p*
Pair 3	Pretest Fear, Trust (Experimental)	3.00	0.43	**2.996**	**93**	**.00****
	Posttest Fear, Trust (Experimental)	2.83	0.28			
Pair 4	Pretest Fear, Trust (Control)	2.69	0.41	-1.244	93	.217
	Posttest Fear, Trust (Control)	2.76	0.36			
Pair 5	Pretest Professional Importance (Experimental)	3.12	0.46	**4.886**	**93**	**.00****
	Posttest Professional Importance (Experimental)	2.87	0.20			
Pair 9	Pretest Mathematics Scale (Experimental)	3.03	0.28	**4.791**	**93**	**.00****
	Posttest Mathematics Scale (Experimental)	2.87	0.16			
Pair 10	Pretest Mathematics Scale (Control)	2.88	0.24	1.449	93	.15
	Posttest Mathematics Scale (Control)	2.83	0.16			

*$p < .05$ ** $p < .01$

When Table 9 is examined, the experimental group's first application of the Mathematics Attitude Scale showed a mean score of $M = 3.03$ ($SD = 0.28$), while the mean score for the control group was $M = 2.88$ ($SD = 0.24$). While the students in the experimental group obtained a mean score of $M = 2.87$ ($SD = 0.17$) from the

second application of the Mathematics Attitude Scale, the students in the control group obtained a mean score of $M = 2.83$ ($SD = 0.16$).

According to the findings, the experimental group students obtained a mean score of $M = 2.87$ in the interest and compassion dimension in the first application of the scale and a mean score of $M = 2.75$ in the second. The control group students in the first application in the same dimension (interest and compassion) obtained a mean score of $M = 2.65$, whilst in the second application it was $M = 2.85$. Therefore, it can be said that there was a low-level increase seen in the attitudes of the control group students about interest and compassion compared to the experimental group.

In terms of occupational materiality, a decrease was observed between the first application ($M = 3.12$) and the second application ($M = 2.87$) mean scores for the experimental group students. In the control group, a decrease was observed between the mean scores of the first ($M = 3.06$) and second ($M = 2.93$) applications. In the dimension of pleasure, the mean scores obtained by the experimental group in the first ($M = 3.04$) and second ($M = 3.04$) applications of the scale did not change, whilst the mean score ($M = 2.83$) obtained in the second application for the control group ($M = 3.13$) was lower than for the first application.

When the findings are examined, it can be seen that the mean score obtained from the second application of the Mathematics Attitude Scale for the experimental and control groups decreased when compared to the first application. In addition, the difference between the second and first applications of the scale for the experimental group was -0.16, whilst for the control group it showed a lesser difference at -0.08. In order to determine whether or not the differences observed between the first and second applications' mean scores of the Mathematics Attitude Scale were statistically significant, comparisons were made between the groups. For this purpose, independent samples *t*-tests were performed for each group. The findings showed that there was no statistically significant difference found between the experimental and control group students' Mathematics Attitude Scale mean scores

for the posttest application ($t(186) = 1.415$; $p > .05$).

However, this finding was not an expected situation. Therefore, the extreme values of the scores obtained by both groups in their posttests were examined using the Mahalanobis test and eliminated from the dataset where applicable. In this context, Figure 3. and Figure 4. illustrate the findings according to the related data.

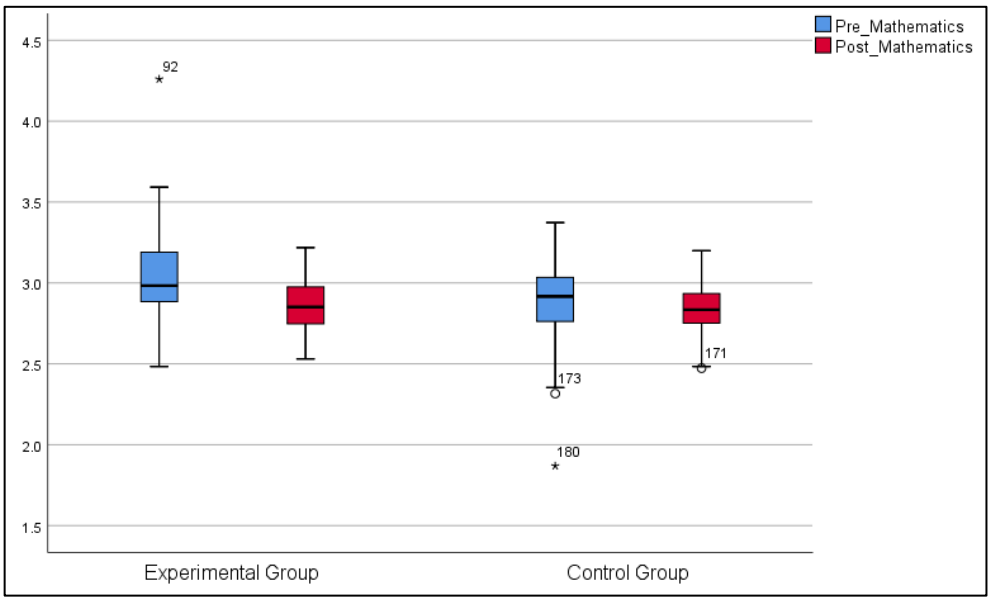


Figure 3. Extreme Values Obtained from Mathematics Attitude Scores

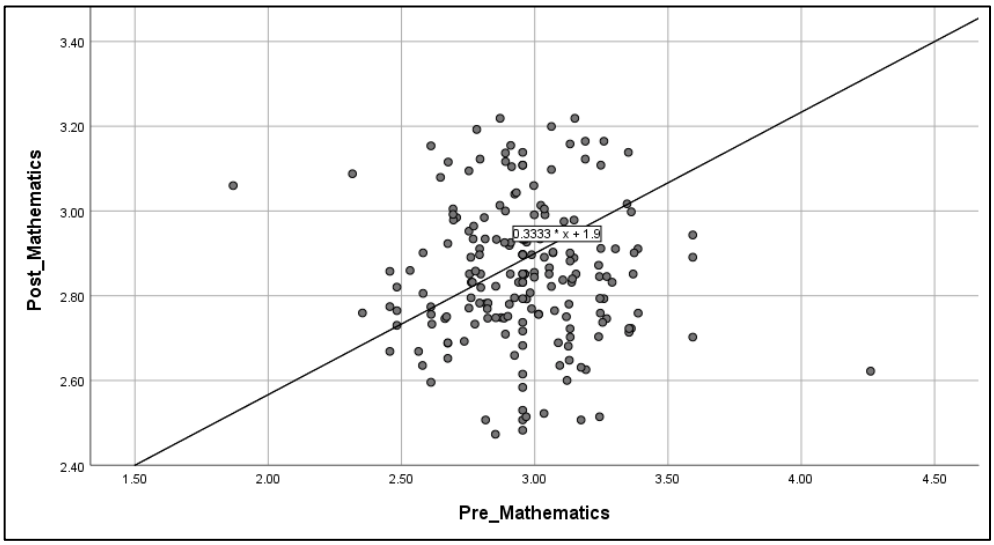


Figure 4. Outliers of Mathematical Attitude Pretest–Posttest Scores

Following the Mahalanobis test, the extreme values of the posttest scores were eliminated from the dataset and the tests then re-conducted. Table 10 presents the results obtained from the independent sample *t*-test analysis to determine whether or not the opinions of the participants about their mathematics attitude differed according to their group.

Table 10. Differences Between Participants' Opinions based on MAS Scores (by Group)

Scale (Test)	Group	*M*	*SD*	*t*	X2-X1	*df*	*p*
Mathematics Attitude Scale (Pretest)	Experimental	3.03	0.28	**3.914**	0.15	**186**	**.00****
	Control	2.88	0.24				
Mathematics Attitude Scale (Posttest)	Experimental	2.88	0.16	1.635	0.41	177	.10
	Control	2.83	0.16				

*$p < .05$ ** $p < .01$ X_2-X_1 = difference in mean value

An independent samples *t*-test was performed to assess whether or not the difference between the MAS mean scores of the participants in the experimental and control groups was statistically significant. The pretest results show that the attitudes of the students in the experimental group towards mathematics showed a significant difference ($t(186) = 3.914$; $p < .00$). In terms of the posttest results, even though the extreme values had been removed, no statistically significant difference was found to exist between the mean scores of the experimental and control groups ($t(177) = 1.635$; $p > .05$).

In addition to the *t*-tests, ANCOVA analysis was conducted in order to observe the mediation effect of the pretest mean scores of both the experimental and control groups' students on their posttest mean scores. The findings obtained from the ANCOVA analysis are presented in Table 11.

Table 11. Covariance Analysis of MAS Pretest–Posttest Scores

Source	SS	*df*	*MS*	*F*	*p*	η_p^2
MAS Pre Test	0.029	1	.029	1.066	.303	.006
Group	0.073	1	.073	2.705	.102	.014
Error	4.969	185	.027			

[a.] R Squared = .016 (Adjusted R Squared = .006)

The findings presented in Table 11 show that the pretest scores did not have any effect on the posttest scores ($F(1.066)$, $p > .05$).

Sobel test analysis was conducted for the purpose of verifying the findings, and also to provide an additional level of analytical detail. Sobel analysis is used in order to observe potential interactions and what effects the independent variable may have by keeping the confidence interval wider. Thus, pretests of the Mathematics Attitude Scale were used as the mediator to investigate the Sobel analysis. In order to test the pretests' mediation effect, a three-stage model was developed, as shown in the conceptual diagram of the model presented as Figure 5.

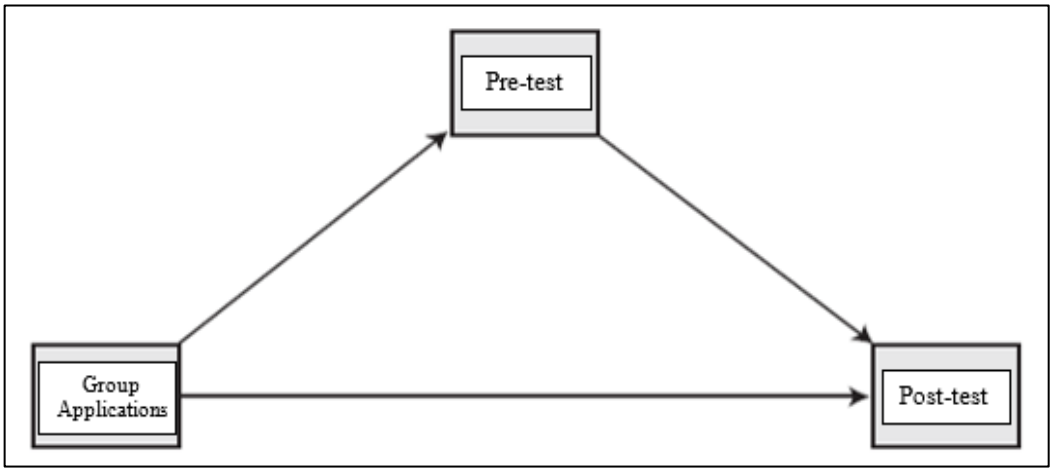


Figure 5. Conceptual Diagram of Multiple Regression Analysis

The effect of the group application on the pretest was examined in the first stage, as can be seen in Table 12. The results clearly show that group activities affected the pretest scores (β = -.15).

Table 12. Effect of Group Applications on Pretest Scores

Model	Unstandardized coefficients		Standardized coefficients	*t*	*p*
	β	*SE*	β		
(Constant)	3.18	.06		53.10	.00*
Group Applications	-0.15	.04	-.15	-3.91	**.00***

[R = .28, R^2 = .08; F = (1, 186) = 15.32, p < .01].

In the second stage, the effect of the group activities on the posttest scores was also examined (see Table 13). The results of the analysis showed that the group activities had no effect on the posttest as the beta value was found to be -0.3 (β = -.03).

Table 13. Effect of Group Activities on Posttest Scores of MAS

Model	Unstandardized coefficients		Standardized coefficients	*t*	*p*
	β	*SE*	β		
(Constant)	2.90	.04		76.76	.00*
Group applications	-0.03	.02	-.03	-1.41	.16

[R= .10, R^2 = .01; F = (1, 186) = 1.53, p > .05]

The mediating role of the pretest was examined in the effect of group activities on posttest scores in the third stage. The findings in Table 14. show that the expected values were reached as a result of the analysis.

Table 14. Effect of Group Applications and Pretest on MAS Posttest Scores

Model	Unstandardized coefficients		Standardized coefficients	*t*	*p*
	β	*SE*	β		
(Constant)	3.05	.15		20.10	.00*
Pretest	-0.05	.05	.27	-1.03	.30
Group Applications	-0.04	.03		-.1.65	.10

[R = .13, R^2 = .02; F = (2, 185) = 1.53, p > .05].

The findings of the Sobel analysis are presented in Table 15.

Table 15. Sobel Analysis Results

Effect	*SE*	*z*	*p*
.0071	.01	.9692	.33

The results showed that there is no mediator effect in the pretest results. However, in order to propose a final decision in this matter, the indirect effect values were examined as this approach may reveal a possible meaningful effect in the wider context. In the model, the total effect of group applications on posttest scores, the direct effect of group applications on posttest scores, and the indirect effect of group applications (with the mediator effect of pretest) on posttest scores was calculated, and results are presented in Table 16.

Table 16. Effect, Direct Effect, and Indirect Effect of Group Applications on MAS Posttest Scores

Total effect: Group applications on Mathematics Attitude Scale posttest scores					
Effect	*SE*	*t*	*p*	LLCI	ULCI
-.0338	.02	-1.41	.16	-.0810	-.0133
Direct effect: Group applications on Mathematics Attitude Scale posttest scores					
Effect	*SE*	*t*	*p*	*LLCI*	*ULCI*
-.0409	.03	-1.65	.10	-.0900	.0082
Indirect effect: Group applications on Mathematics Attitude Scale posttest scores (with mediator effect of pretest)					
	Effect	*SE*	*BootLLCI*	*BootULCI*	
Pretest	.0071	.01	-.0062	-.0256	

The findings presented in Table 16. show the mediating effect of the pretest scores to be insufficient to explain the effect of group applications on Mathematics Attitude Scale posttest scores (BootLLCI – BootULCI). Therefore, it can be clearly stated that it is hard to talk about effect in the practical sense, because the effect size is shown to be low and the mediating effect of the pretest scores is very small compared to the effect of group activities on the posttest scores.

4.1.2 Research Question B: Computational Thinking Skills Self-Efficacy

This subsection presents the findings in response to the second research sub-question (b): *"Is there a significant difference between the control and experimental groups according to their Computational Thinking Skills Self-Efficacy Scale scores?"* The hypothesis for the second research questions is stated below:

H_0: Integrating mathematics into programming courses increases student's computational thinking skills.

There is a significant difference between the experimental and control groups to their computational thinking skills.

H_a: Integrating mathematics into programming courses does not increase student's computational thinking skills.

There is no significant difference between the experimental and control groups to their computational thinking skills.

First, the scores of the Computational Thinking Skills Self-Efficacy Scale were examined, having been obtained from both the experimental and control student groups. In order to determine whether or not a statistically significant difference exists between the mean scores from the pretest and posttest applications of the scale for both groups, a paired-samples *t*-test was performed. The procedure was then followed by independent samples *t*-tests, so as to compare the differences between the pretest and posttest scores of the experimental and control groups. Table 17. presents the results of the students' Computational Thinking Skills Self-Efficacy Scale for the experimental and control groups. The results are shown according to the pretest and posttest scale applications.

Table 17. Mean Scores for Paired Groups t-Test of Students' CTSSES (Pretest–Posttest)

Pair	Test /Dimension (Group)	M	SD	t	df	p
Pair 1	Pretest / Algorithm Design (Experimental)	8.84	4.16	**-11.485**	**93**	**.00****
	Posttest / Algorithm Design (Experimental)	14.94	3.16			
Pair 2	Pretest / Algorithm Design (Control)	10.39	4.96	0.965	93	.34
	Posttest / Algorithm Design (Control)	9.80	3.92			

Table 17. Mean Scores for Paired Groups t-Test of Students' Computational Thinking Skills Self-Efficacy Scores (Pretest–Posttest) (continued)

Pair	Test /Dimension (Group)	M	SD	t	df	p
Pair 3	Pretest / Problem Solving (Experimental)	12.78	4.44	**-4.54**	**93**	**.00****
	Posttest / Problem Solving (Experimental)	15.47	3.74			
Pair 4	Pretest / Problem Solving (Control)	14.29	4.38	1.882	93	.06
	Posttest / Problem Solving (Control)	13.16	4.16			
Pair 5	Pretest / Data Processing (Experimental)	8.62	3.65	**-6.232**	**93**	**.00****
	Posttest / Data Processing (Experimental)	11.54	2.44			
Pair 6	Pretest / Data Processing (Control)	9.04	4.15	**2.935**	**93**	**.00***
	Posttest / Data Processing (Control)	7.51	2.75			
Pair 7	Pretest / Basic Programming (Experimental)	3.35	2.76	**-14.59**	**93**	**.00****
	Posttest / Basic Programming (Experimental)	8.51	1.92			
Pair 8	Pretest / Basic Programming (Control)	4.36	3.38	**-3.082**	**93**	**.00****
	Posttest / Basic Programming (Control)	5.67	2.12			
Pair 9	Pretest / Self Confidence (Experimental)	6.40	2.71	**-5.824**	**93**	**.00****
	Posttest / Self Confidence (Experimental)	8.29	1.80			
Pair 10	Pretest / Whole Scale (Experimental)	40.29	13.81	**-9.775**	**93**	**.00****
	Posttest / Whole Scale (Experimental)	58.74	11.87			
Pair 11	Pretest / Whole Scale (Control)	44.57	16.11	0.595	93	.55
	Posttest / Whole Scale (Control)	43.31	13.51			

When Table 17. is examined, it can be seen that the students in the experimental group achieved a mean score average of $M = 40.29$ ($SD = 13.81$) from the pretest

application of the Computational Thinking Skills Self-Efficacy Scale, whilst the mean score average of the students in the control group was $M = 44.57$ ($SD = 16.11$). Similarly, the posttests result of the experimental group students was $M = 58.74$ ($SD = 11.87$), whilst the control group students' mean scores averaged $M = 43.31$ ($SD = 13.51$). When these findings are compared, it can clearly be seen that the students in the experimental groups achieved a higher mean score average compared to the students from the control groups. Also, the students in the control groups showed a slightly lower average mean score in the posttest application of the Computational Thinking Skills Self-Efficacy Scale when compared to the pretest application.

In addition to the whole scale, results for each subscale (dimension) were also examined. The experimental group students can be seen to have obtained higher average scores at the end of the experimental process (posttest) regarding design algorithms (8.84 to 14.94), problem solving (12.78 to 15.47), data processing (8.62 to 11.54), basic programming (3.35 to 8.51), and self-confidence (6.40 to 8.29). However, students in the control groups showed a decrease across some of the subscales, with lower scores obtained in the posttest compared to the pretest application for algorithm design (10.39 to 9.80), problem solving (14.29 to 13.16), and data processing (9.04 to 7.51). On the other hand, the control groups' students increased their averages slightly in the posttest compared to the pretest for basic programming (4.36 to 5.67) and also the self-confidence (6.56 to 7.17) dimension.

These findings show that the average mean score increase achieved by the students from the posttest application concerning the whole scale for the experimental group was +18.45, which represents a significant increase, whilst the average mean score of the control group showed a slight -1.26 decrease. Therefore, the findings obtained from the paired *t*-test show a statistically significant difference between the mean scores obtained from the pretest and posttest applications of the Computational Thinking Skills Self-Efficacy Scale for the experimental group ($t(93) = -9,775$; $p < .01$). In other words, the experimental groups' students' self-efficacy skills

concerning information processing (computational) thinking skills showed a significant improvement following application of the experimental process when compared to the pre-experimental application of the scale. There was no statistically significant difference found to exist between the control groups' students' averages from the pretest and posttest applications of the same scale. Based on this finding, it can be said that the control groups' students' self-efficacy concerning their information processing (computational) thinking skills did not differ during the application.

Extreme values of the scale's scores from both student groups in the posttests were examined and eliminated from the dataset. In this context, Figure 6. and Figure 7. illustrate this process.

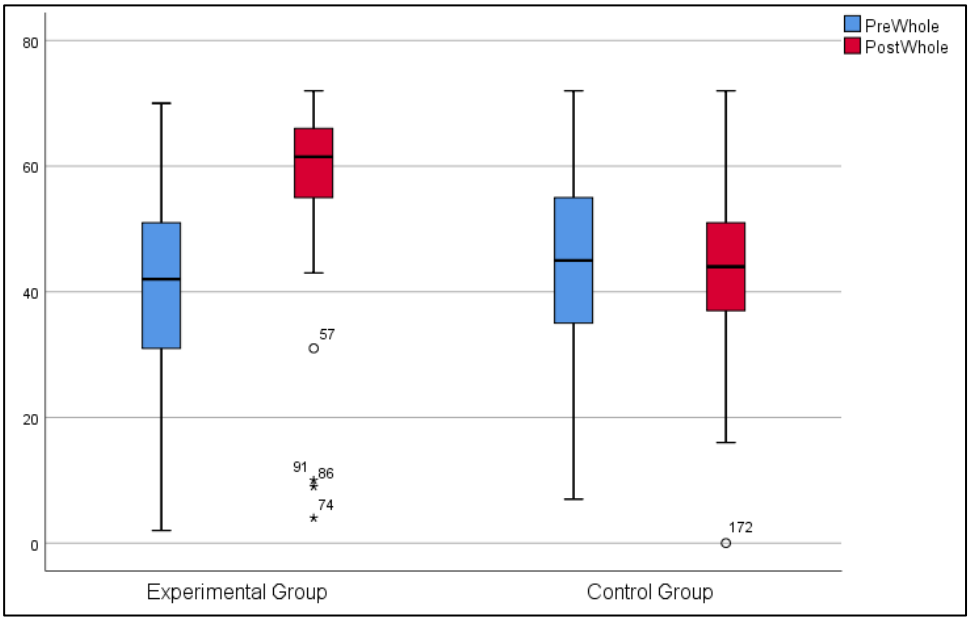


Figure 6. Extreme Values from Computational Thinking Skills Self-Efficacy Scale

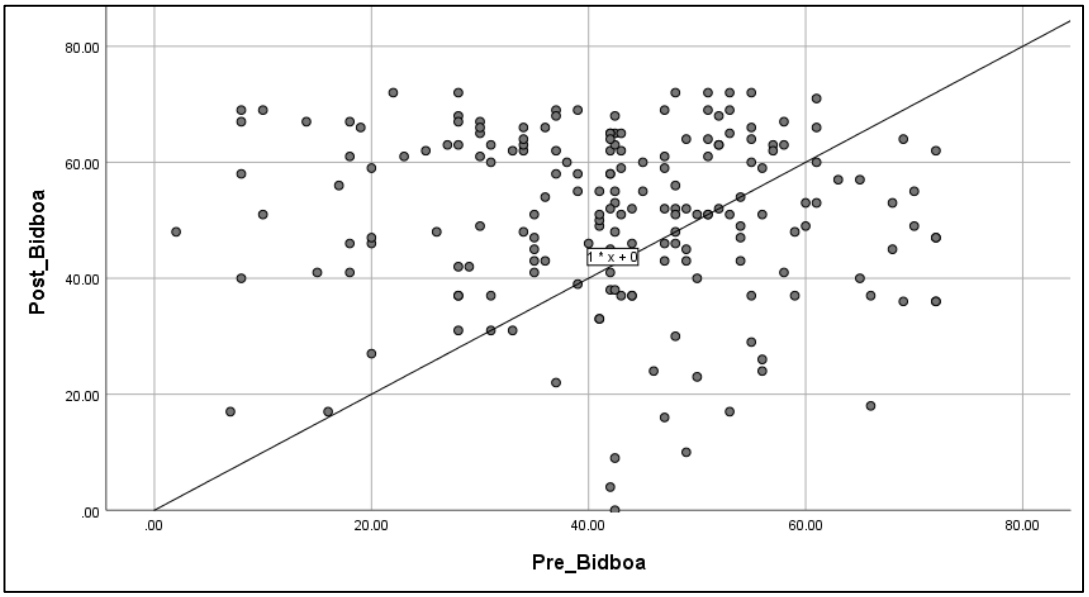


Figure 7. Outliers of Computational Thinking Skills Self-Efficacy Scale (Pretest–Posttest)

After eliminating the extreme values of the dataset from the posttest scores, the independent sample *t*-test analysis was conducted in order to see if there was any difference between the experimental and control groups in respect to their Computational Thinking Self-Efficacy Skills Scale scores. The results of the independent sample *t*-test analysis are presented in Table 18.

Table 18. Between Participants' Opinions Regarding Subscales of CTSSES Scores (by Group)

Scale (Test)	Group	*M*	*SD*	*t*	X2-X1	*df*	*p*
Computational Thinking Skills Self-Efficacy Scale (Pretest)	Experimental	40.29	13.81	**-1.96**	**-4.28**	186	**.05***
	Control	44.57	16.11				
Computational Thinking Skills Self-Efficacy Scale (Posttest)	Experimental	58.74	11.87	**10.99**	**17.44**	182	**.00****
	Control	43.31	13.51				

*$p < .05$ ** $p < .01$ X_2-X_1 = difference in mean value

In order to determine whether or not the difference between the mean scores of the experimental and control groups' students on the Computational Thinking Skills Self-Efficacy Scale were statistically significant, an independent samples *t*-test test was performed. At the end of the test, it was observed that the students' computational thinking skills self-efficacy differed significantly between the pretest ($t(186) = -1.96$; $p < .05$) and posttest ($t(186) = 10.99$; $p < .01$) scale applications.

While the mean scores from the Computational Thinking Skills Self-Efficacy Scale for the experimental groups' students was lower than the from the control groups prior to the experimental process (pretest), it was found to be higher after the experimental process had been applied (posttest). Based on this finding, it can be said that the teaching of coding with an IA increases students' self-efficacy in computational thinking skills.

In addition to the *t*-tests, ANCOVA analysis was applied in order to reveal any effect that the pretest averages of the experimental and control groups' students had on the posttest averages. The results of this analysis are presented in Table 19.

Table 19. Covariance Analysis CTSSES Pretest–Posttest Scores

Source	*SS*	*df*	*MS*	*F*	*p*	η_p^2
CTSSES Pre-Test	27.081	1	27.081	.233	.630	.001
Group	13,887.309	1	13,887.309	119.357	.000	.397
Error	21,059.595	181	116.351			

[a.] R Squared = .400 (Adjusted R Squared = .393)

The covariance analysis findings showed that the students' pretest scores did not influence their posttest scores ($F(1, 181) = .233$; $p > .05$). In other words, while the pretest has no effect on the posttest, it can be said that the Computational Thinking Skills Self-Efficacy Scale averages scores for the experimental groups' students increased significantly in the posttest application when compared to the control groups' students.

4.1.3 Research Question C: Computer Programming Self-Efficacy

This subsection presents the findings in response to the third research sub-question (c): *"Is there a significant difference between the control and experimental groups according to their Computer Programming Self-Efficacy Scale scores?"*

The hypothesis for the third research questions is stated below:

> H_0: Integrating mathematics into programming courses increases student's programming skills.
>
> There is a significant difference between the experimental and control groups to their programming skills.
>
> H_a: Integrating mathematics into programming courses does not increase student's programming skills.
>
> There is no significant difference between the experimental and control groups to their programming skills.

First, the average scores from the applications of the Computer Programming Self-Efficacy Scale (CPSES) were examined for both the experimental and control groups. In order to establish whether or not there was a statistically significant difference between the experimental and control groups' students, the averages of from both the pretest and posttest applications were analyzed by a paired samples *t*-test. This procedure was applied both to the experimental and the control groups. Second, independent sample *t*-tests were also performed for both the experimental and control groups in order to compare the pretest and posttest scores.

The results of the experimental and control group students' averages obtained from the pretest and posttest application of the Computer Programming Self-Efficacy Scale are presented in Table 20.

Table 20. Group t-Test: Difference between Students' CPSES Scores (Pretest–Posttest)

Pair	Test /Dimension (Group)	*M*	*SD*	*t*	*df*	*p*
Pair 1	Pretest / Basic Coding (Experimental)	3.26	1.49	**-15.809**	**93**	**.00**
	Posttest / Basic Coding (Experimental)	6.06	0.88			
Pair 2	Pretest / Basic Coding (Control)	4.44	1.67	-1.672	93	.10
	Posttest / Basic Coding (Control)	4.76	1.18			
Pair 3	Pretest / Complex Coding (Experimental)	2.79	1.19	**-13.326**	**93**	**.00**
	Posttest / Complex Coding (Experimental)	5.07	1.05			
Pair 4	Pretest / Complex Coding (Control)	3.73	1.50	**2.331**	**93**	**.02**
	Posttest / Complex Coding (Control)	3.26	1.30			
Pair 5	Pretest / Whole Scale (Experimental)	6.05	2.38	**-16.378**	**93**	**.00**
	Posttest / Whole Scale (Experimental)	11.14	1.71			
Pair 6	Pretest / Whole Scale (Control)	8.16	2.97	0.417	93	.68
	Posttest / Whole Scale (Control)	8.01	2.27			

When the results from Table 20. are examined, it can be seen that the average Computer Programming Self-Efficacy Scale score for the experimental groups' pretests was $M = 6.05$ ($SD = 2.38$), whilst for the control groups' pretests it was $M = 8.16$ ($SD = 2.97$). The average posttest score for the experimental groups was $M = 11.14$ ($SD = 1.71$), whilst it was $M = 8.01$ ($SD = 2.27$) for the control groups.

The findings from the components (dimensions) of the scale show that the experimental groups' students obtained higher average scores from the Computer Programming Self-Efficacy Scale at the end of the experimental study process (posttest). The results for the control groups showed slightly lower average posttest values than for the pretest application.

When the component dimensions of the scale were examined, the experimental groups' students showed higher average scores in their posttest results when compared to the pretest for basic coding skills (3.26 to 6.06) and complex coding skills (2.79 to 5.07). The averages of the control groups' students for basic coding skills increased slightly (4.44 to 4.76) in the posttest, but achieved lower average scores for complex coding skills (3.73 to 3.26). Therefore, it can clearly be seen that the average scores achieved in the posttests by the experimental groups' students increased when compared to their pretest scores. However, the averages of the control groups mostly decreased when the pretest application results were compared to that of their posttest. In addition, the difference between the pretest and posttest scale applications for the experimental groups was +5.09, whilst it was -0.15 for the control groups.

The paired *t*-test results show that the pretest to posttest differences in scores for the Computer Programming Self-Efficacy Scale between the experimental and control groups were found to be statistically significant ($t(93) = -16.38$; $p < .01$). In other words, the programming skills of the experimental groups' students showed a significant improvement following application of the experimental process (posttest) when compared to the pre-experimental process (pretest). However, for the control groups, no statistically significant difference was observed between the averages obtained from the pretest and posttest applications of the Computer Programming Self-Efficacy Scale.

Prior to conducting the between group comparison, extreme values of the scores obtained by both sets of groups in the pretests and posttests were examined and

eliminated from the dataset. In this context, Figure 8. and Figure 9. were created according to the related data.

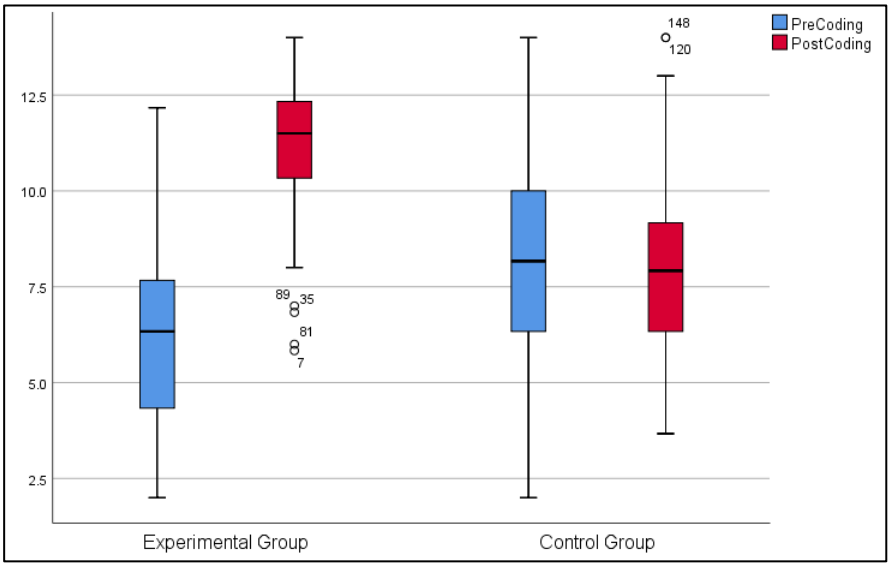


Figure 8. Extreme Values from Computer Programming Self-Efficacy Scale

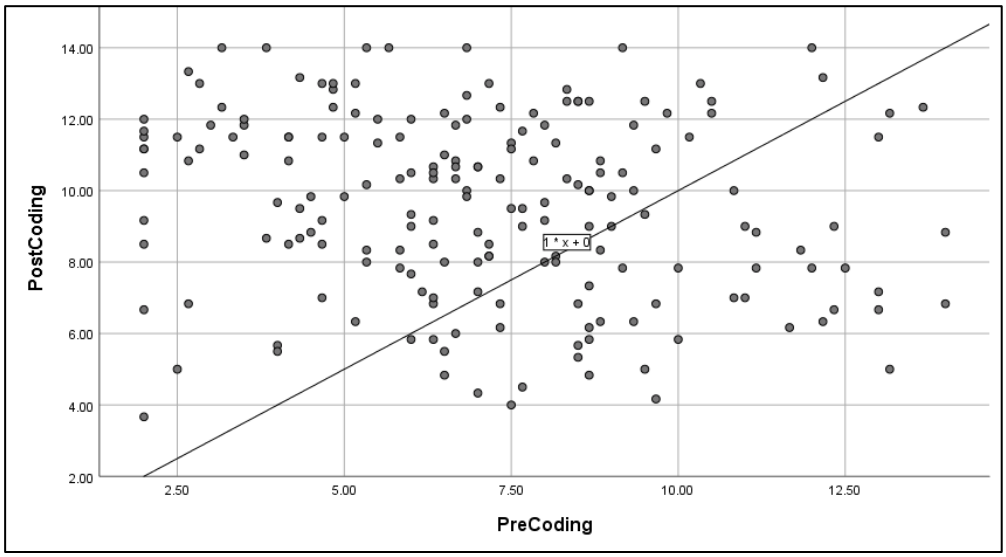


Figure 9. Outliers of Computer Programming Self-Efficacy Scale Pretest–Posttest Applications

The extreme posttest score values were eliminated from the dataset. Table 21. presents the results obtained from the independent sample *t*-test analysis which was

conducted in order to determine whether or not the opinions of the participants on the subject of their programming skills differed according to their group.

Table 21. Participants' Opinions Regarding CPSES Scores (by Group)

Scale (Test)	Group	*M*	*SD*	*t*	X2-X1	*df*	*p*
Computer Programming Self-Efficacy (Pretest)	Experimental	6.05	2.38	**-5.387**	**-2.12**	186	**.00****
	Control	8.16	2.97				
Computer Programming Self-Efficacy (Posttest)	Experimental	11.35	1.41	**12.98**	**3.47**	180	**.00****
	Control	7.88	2.11				

*$p < .05$ ** $p < .01$ X_2-X_1 = difference in mean value

In order to establish whether or not there was a significant difference between the mean scores of the experimental and control groups' students for the Computer Programming Self-Efficacy Scale, an independent samples *t*-test test was performed. The test results show that there was a statistically significant difference found between the experimental and the control groups in terms of their pretest scores ($t(186) = -5.39$; $p < .01$) and posttest scores ($t(180) = 12.98$; $p < .01$).

While the mean scores of the Computer Programming Self-Efficacy Scale for the experimental groups' students were lower than those of the control groups prior to the experimental process application (pretest), they were shown to be higher at the end of the study according to their posttest results. Based on these findings, it can be said that the teaching of programming with an IA increases students' programming skills to a level that is statistically significant.

Next, ANCOVA analysis was conducted in order to observe the effect of the pretests average scores of the groups on the posttests average scores. The results from this analysis are presented in Table 22.

Table 22. Covariance Analysis of Students' CPSES Pretest-Posttest Scores

Source	*SS*	*df*	*MS*	*F*	*p*	η_p^2
CPSES Pre-Test	1.916	1	1.916	.589	.444	.003
Group	496.040	1	496.040	152.581	.000	.460
Error	581.928	179	3.251			

[a.] R Squared = .485 (Adjusted R Squared = .479)

The findings showed that the pretest scores did not have a statistically significant effect on the posttest ($F(1, 179) = 152.58$; $p > .05$). In other words, while the pretest had no effect on the posttest, it can be said that the programming skills averages of the experimental groups increased significantly in the posttest application when compared to those of the control groups.

4.1.4 Research Question D: Small Basic Programming Achievement

This subsection presents the findings in response to the fourth research sub-question (d): *"Is there a significant difference between the control and experimental groups in their retention of Small Basic Programming Achievement Test results?"*

The hypothesis for the fourth research questions is stated below:

> H_0: Integrating mathematics into programming courses increases student's reintetion of programming skills.
>
> There is a significant difference between the experimental and control groups to their reintetion of programming skills.
>
> H_a: Integrating mathematics into programming courses does not increase student's reintetion of programming skills.
>
> There is a significant difference between the experimental and control groups to their reintetion of programming skills.

In analyzing the collected data to answer the fourth research sub-question, the same procedure was followed as applied to the first three research sub-questions. First, the BASIC Programming Achievement Test was applied twice and the scores analyzed using paired-samples *t*-test in order to determine if there was a significant difference found between the averages obtained from the pretest and posttest applications to students from the experimental and control groups. After that, repeated measures variance analysis was performed to compare the pretest and posttest scores of the experimental and control groups. Extreme values of the scores obtained from the pretest and posttest applications of the BASIC Programming Achievement Test for the experimental and control groups were examined and eliminated from the dataset prior to conducting in group comparison. Extreme values for both groups are illustrated as shown in Figure 10. and Figure 11.

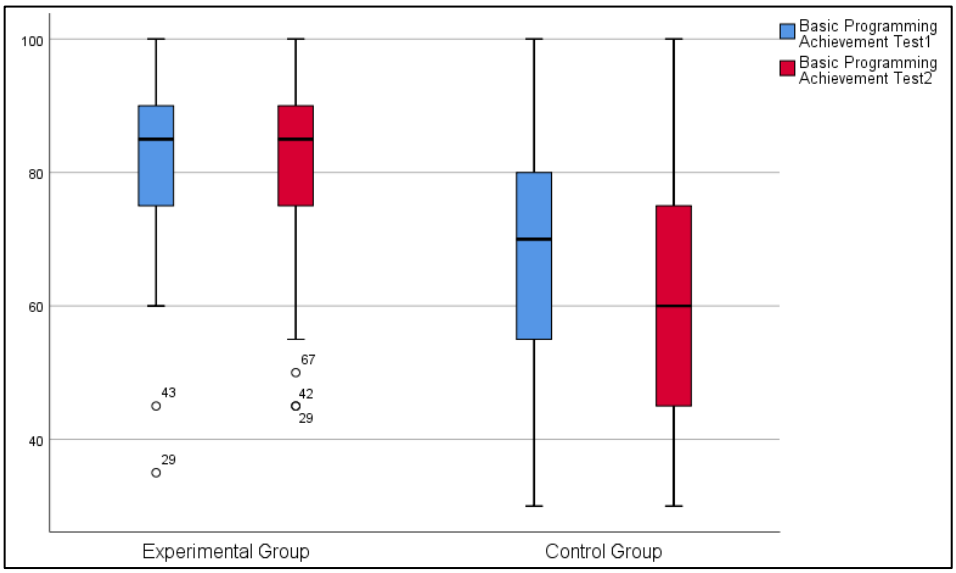


Figure 10. Extreme Values from Small Basic Programming Achievement Tests

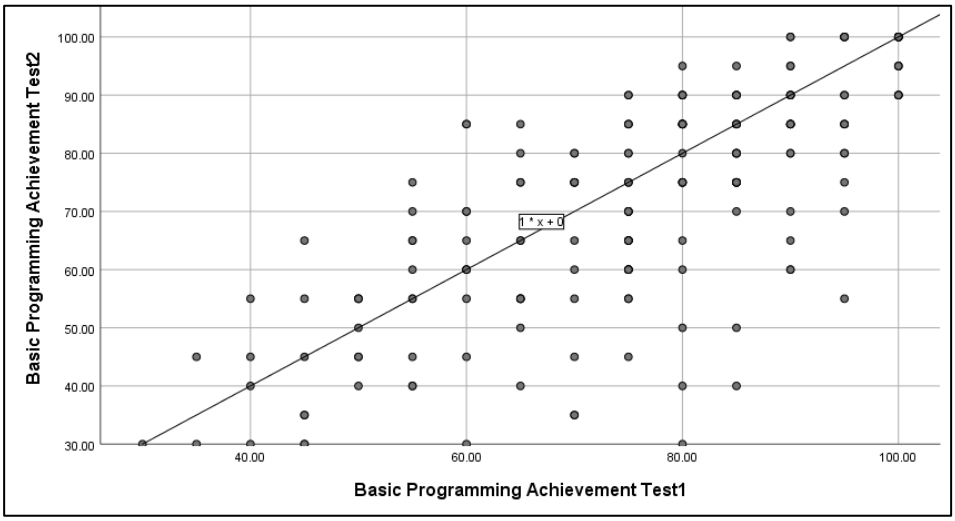


Figure 11. Outliers of Small Basic Programming Achievement Test (Pretest–Posttest)

The results of the experimental and control groups' students' average scores obtained from the pretest and the posttest applications of the BASIC Programming Achievement Test are presented in Table 23.

Table 23. Paired Groups t-Test: Differences Between Mean Scores from Pretest & Posttest of SBPAT

Pair	Test (Group)	*M*	*SD*	*t*	*df*	*p*
Pair 1	Pretest / SBPAT (Experimental)	83.32	12.04	0.478	90	.62
	Posttest / SBPAT (Experimental)	84.28	11.58			
Pair 2	Pretest SBPAT (Control)	67.98	16.53	**5.144**	**93**	**.00**
	Posttest / SBPAT (Control)	60.32	18.03			

When Table 23. is examined, it can be seen that the students in the experimental group obtained an average mean score of $M = 83.32$ from the pretest application of the Small Basic Programming Achievement Test, whilst the control groups' students obtained an average mean score of $M = 67.98$. The average mean scores from the posttests were $M = 84.28$ from the experimental groups' students, and $M = 60.32$ for

students of the control groups. The mean scores achieved by the experimental groups' students increased slightly by +0.96 points in the posttest. However, for the control groups' students, their posttest Small Basic Programming Achievement Test results were much lower, representing a decrease of -7.66.

The paired *t*-test was used to determine whether or not the changes seen in the mean scores were statistically significant. The findings of the paired-samples *t*-test showed that there was no statistically significant difference between the pretest and posttest Small Basic Programming Achievement Test scores for students in the experimental group, but that there was a statistically significant difference ($p < .01$) found to exist between the mean scores average obtained from the pretest and posttest results for the control groups' students. According to Table 23, the average mean score of the control groups' students in the posttest application of the Small Basic Programming Achievement Test differed to a statistically significant level from that of the pretest ($t(93) = 5.144$; $p < .01$). Based on this finding, it can be said that the experimental groups' students did not experience a decrease in the average score in contrast to the control group students. In fact, the achievement test scores of the experimental group students increased slightly (+0.96), although the increase was not found to be statistically significant. Also, the findings show that posttest scores of the Small Basic Programming Achievement Test in terms of the control groups' students were lower (-7.66) than the pretest scores.

ANCOVA (repeated measures variance analysis) was applied in order to examine whether or not the mean distribution differed according to the sub-groups. In the analysis, the scores obtained from the pretest and posttest applications of the Small Basic Programming Achievement Test were compared in eight groups, four of which were experimental groups and four control groups. According to the findings from the ANCOVA analysis, there was a difference observed between the average scores of the experimental and control groups pretest and posttest scores for the Small Basic Programming Achievement Test, and that the difference was shown to be statistically

significant (Wilks Lambda = .915, F = 19.669, p < .001). Figure 12. graphically illustrates this finding.

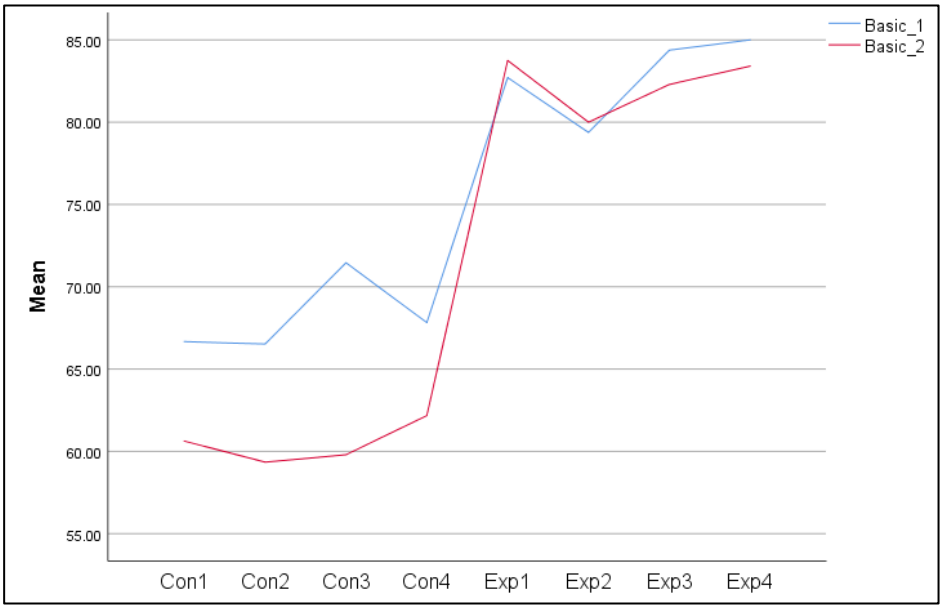


Figure 12. Differences Between Groups for Students' Small Basic Programming Achievement Test Pretest–Posttest Scores

The graph illustrated in Figure 12 shows that the Small Basic Programming Achievement Test pretest and posttest averages of the four experimental groups were close to each other. However, when the average scores from the four control groups were examined, it can be seen that the scores decreased significantly between the pretest and posttest applications of the Small Basic Programming Achievement Test.

Independent samples *t*-tests were then conducted so as to observe how the difference compared between the primary groups (experimental and control). The findings from this test are presented in Table 24.

Table 24. Differences Between Participants' SBPAT Scores (by Group)

Test	Group	*M*	*SD*	*t*	X2-X1	*df*	*p*
SBPAT (Pretest)	Experimental	83.32	12.04	**6.82**	**.96**	184	**.00****
	Control	84.28	11.58				
SBPAT (Posttest)	Experimental	67.98	16.53	**10.52**	**-7.66**	181	**.00****
	Control	60.32	18.03				

The independent samples *t*-test was performed in order to determine if there was a significant difference between the mean scores of the experimental and control groups for the Small Basic Programming Achievement Test results. According to the *t*-test results, the pretest ($t(184) = 6.82$; $p < .01$) and posttest ($t(181) = 10.52$; $p < .01$) results were found to be statistically different. Based on these findings, it can be said that students who were taught programming with the integration of the IA achieved higher results than the students taught programming according to the traditional single-subject method. Figure 13 graphically illustrates the findings from this test.

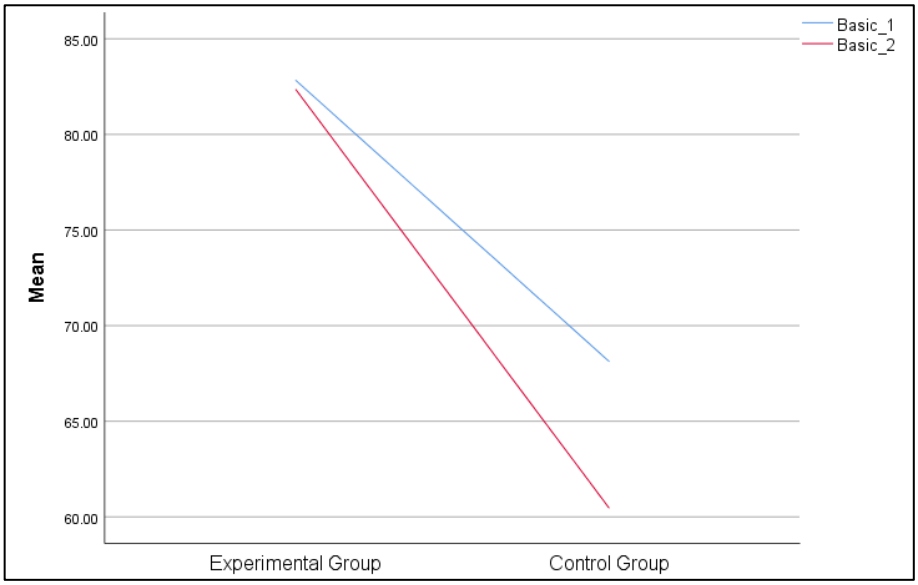


Figure 13 Differences Between Groups for Students' Small Basic Programming Achievement (Pretest–Posttest) Scores

4.2 Qualitative Analysis Findings

This section details how the qualitative data of the study were analyzed in order to answer the two research sub-questions that related to the interviews conducted with the participant students and teachers. In total, 24 students' interviews were conducted, plus both of the teachers who taught classes as part of the study were also interviewed. For the student interviews, three students from each group were chosen according to their mathematics GPA level, with one student who held the highest GPA, one student whose GPA was ranked in the middle of the class, and one student who held the lowest GPA in the class. Verbal data were obtained from the 26 interviews.

4.2.1 Descriptive Analysis of Qualitative Data

The verbal data obtained from the interviews were formed as codes in accordance with the content analysis method. The codes were generated and then arranged under themes according to their characteristic similarities. The 24 interviews conducted with the students were divided into the following five themes;

- Programming language tools used prior to the study,
- Opinions regarding challenges using MS Small Basic programming tool,
- Opinions regarding learning programming with MS Small Basic programming tool,
- Opinions regarding the most helpful elements during learning programming,
- Positive and negative opinions about the MS Small Basic programming tool.

These five main themes were formed based on the data collected from both experimental and control group students. Together with the aforementioned student levels (Level 1: highest GPA, Level 2: middle GPA, Level 3: lowest GPA), representing the students' mathematics GPA results, Figure 14 provides an illustrative summary of the content analysis findings.

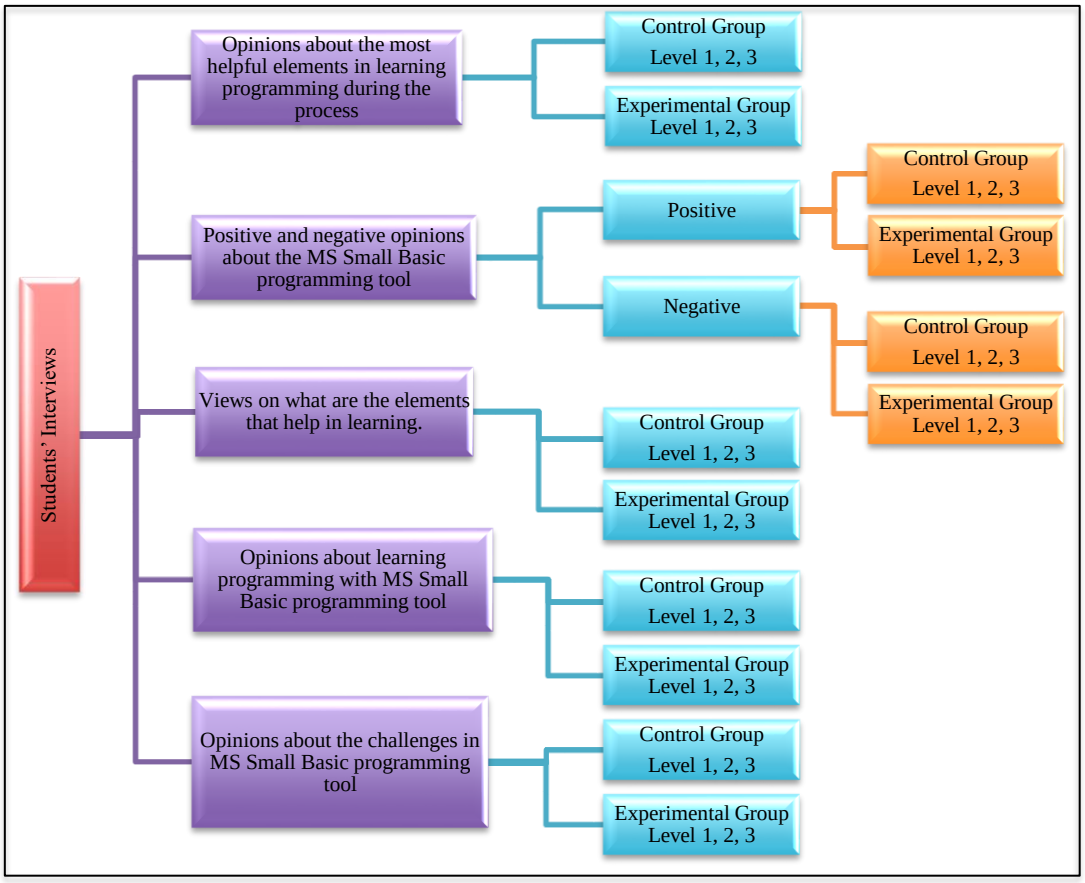


Figure 14. *Themes and Sub-themes of the Study*

As shown in Figure 14., the views of the interviewed experimental and control group students on the teaching of programming using Small Basic based on an IA were examined according to their mathematics academic achievement (mathematics GPA), and divided into three student levels by the researcher. These three student levels were formed in accordance with the students' mathematics GPA, which was calculated based on their cumulative mathematics scores during Grade 5, Grade 6, and Grade 7. Level 1 was assigned to the students with the highest mathematics GPA score in their class, whilst Level 2 was assigned to the students with the middle-ranking mathematics GPA score, and Level 3 to the students with the lowest mathematics GPA score in their class. The responses of the students to the predefined interview questions were examined thematically in accordance with their assigned levels, as well as to which group they belonged, experimental or control. Information

about the 24 students interviewed, and the codes they were assigned by the researcher, are presented in Table 25.

Table 25. Details and Assigned Codes for Interviewed Students

Order	ID	Gender	Group	Level
Student 1	SE1	F	EXP1	1
Student 2	SE2	M	EXP1	2
Student 3	SE3	M	EXP1	3
Student 4	SC1	M	CG1	1
Student 5	SC2	M	CG1	2
Student 6	SC3	F	CG1	3
Student 7	SE4	M	EXP2	1
Student 8	SE5	F	EXP2	2
Student 9	SE6	F	EXP2	3
Student 10	SC4	F	CG2	1
Student 11	SC5	M	CG2	2
Student 12	SC6	M	CG2	3
Student 13	SE7	F	EXP3	1
Student 14	SE8	M	EXP3	2
Student 15	SE9	F	EXP3	3
Student 16	SC7	F	CG3	1
Student 17	SC8	M	CG3	2
Student 18	SC9	F	CG3	3
Student 19	SE10	F	EXP4	1
Student 20	SE11	F	EXP4	2
Student 21	SE12	M	EXP4	3
Student 22	SC10	M	CG4	1
Student 23	SC11	F	CG4	2
Student 24	SC12	M	CG4	3

Findings from a preliminary examination of the interview transcripts were assessed in order to ascertain whether or not there were similarities between the opinions given by the 24 interviewed students. In this context, the interview data were examined and similarities established using Pearson Word Correlation calculation in the Nvivo 12 program. The similarities were grouped according to the most common findings, as can be seen in Figure 14.

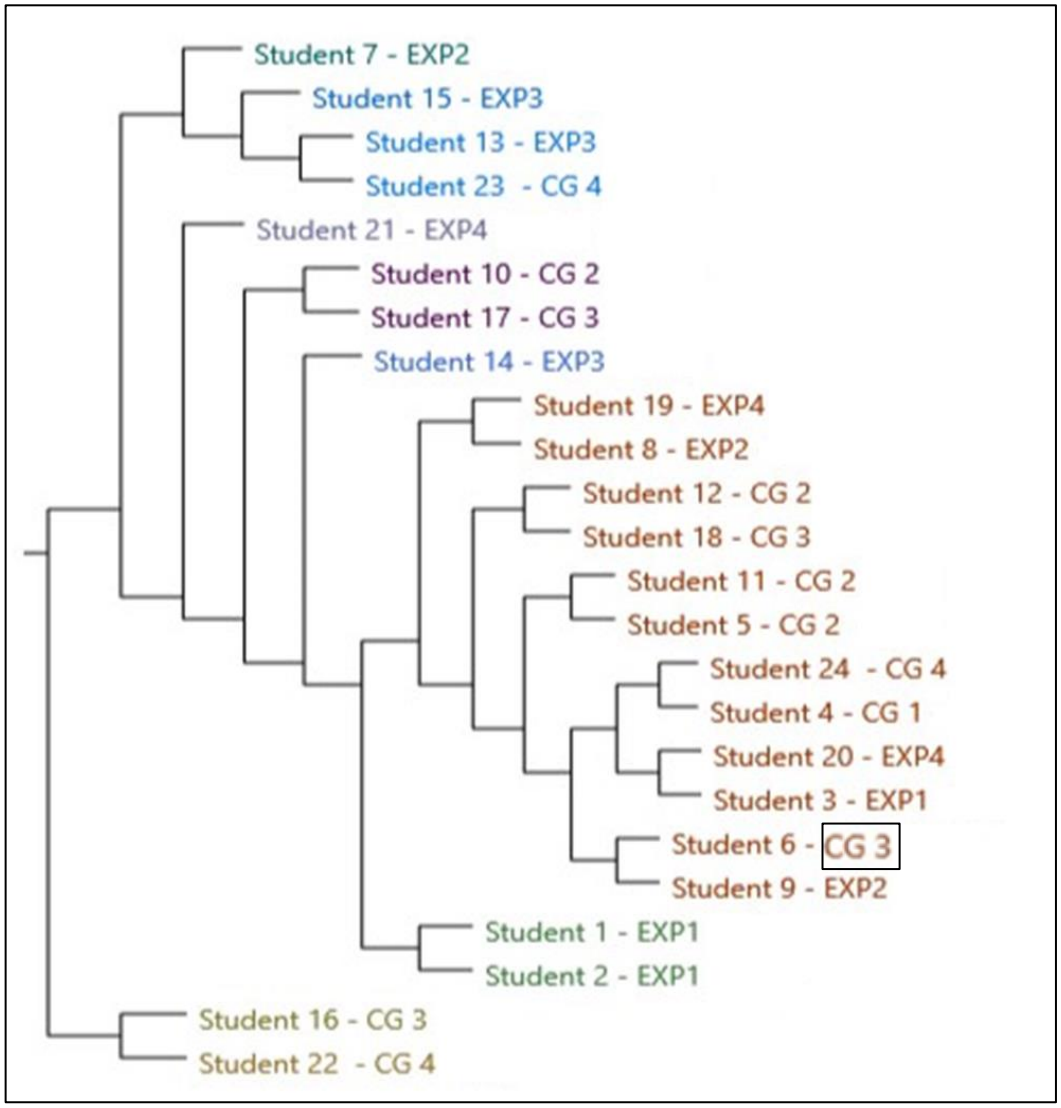


Figure 15. Similarities Between Students' Views (by Group)

When Figure 15. is examined, it can be seen that the opinions of the students from both the experimental and control groups were found to be generally similar.

However, it was observed that some students from one group expressed similar opinions to students in the opposite groups (e.g., experimental group students having expressed similar opinions to control group students). These similarities are paired in the same column such as Student 13 EXP3 and Student 23 CG4, or Student 6 CG and Student 9 EXP2.

The interviews conducted with the 24 students were based on seven predefined questions. The first two questions asked concerned the participants' characteristics. The following five questions were asked in order to determine the level of the students' programming knowledge prior to the study, their view about the MS Small Basic Programming Tool, with both positive and negative responses recorded, as well as their experiences and perceived challenges learning about the program and how to code with it.

4.2.2 Research Question E: Teachers' Thoughts on Teaching Visual Programming Tool

This subsection presents the findings in response to the fifth research sub-question (e): *"What do teachers think about the teaching of visual programming tool using traditional and interdisciplinary approach methods?"*

The verbal data collected from the interviews conducted with the two participant teachers were analyzed by content analysis. The interviewed teachers were both ICT teachers; with Teacher 1 (T1) having taught information and computer technology classes for 6 years, and Teacher 2 (T2) having taught for 12 years. A code table was constructed according to the interview data, as shown in Table 26.

Table 26. Teachers' Views on Programming Teaching

Problem Solving	Computational Thinking	Algorithmic Thinking
Love of mathematics and coding success are directly proportional	Mentoring supports similar capabilities in programming & mathematics	Online assessment
Abstract problems	Programming projects are functional	Setting up an equation
Real-life samples	Inductive teaching	Concretizing abstract subjects
Cycle-variable	HTML	CSS
Action script	Block-coding language	Small Basic

Both T1 and T2 have similar teaching experience with programming languages, and T1 stated these languages as being *"HTML, CSS, Python, Small Basic, block-coding languages,* [and] *action script,"* whilst T2 indicated more general thoughts instead of giving programming language names, saying: *"We are trying to mentor in the teaching of languages that may be needed in different branches in the future."*

When the teachers were asked for the reasons for teaching programming languages to young students, T1 briefly said the purpose was *"To make students learn how to write program code,"* while T2 stated *"I teach programming languages to help students improve their algorithmic thinking, problem solving, and computational thinking skills, and because mentoring students in these three areas is important during early ages."*

The teachers were asked whether or not they experienced any difficulties whilst teaching computer science and programming to young students. T1 stated having had no difficulty in teaching programming languages to the students, whilst T2 sometimes stated sometimes having had difficulty finding the right subjects, saying that *"Sometimes I find it hard to find an interdisciplinary subject that will complement the work."*

The next question asked to the teachers was whether or not they considered programming instruction integrated with a mathematics course had the effect of

increasing students' motivation to learn. Both of the teachers answered "yes" to this question. Also, the teachers shared their opinions about integrating mathematics as a secondary discipline on a programming course. T1 stated that, *"I think that using mathematical elements increases motivation while learning programming and supporting mathematics, and in similar abilities,"* whilst T2's opinion was that, *"Of course, it has had a positive effect for math lovers, but for those who do not* [love math]*, it can have a negative effect."*

The third question directed to the teachers sought their views on the differences between teaching programming through integration with mathematics and the classical method of having distinctly separate subjects. The teachers expressed both similar and differing views on this subject. T1 stated there being certain advantages to integrating mathematics as a secondary discipline in programming courses, saying that *"Since the project is based on clearer and more accurate conclusions and judgments, I think that making programming projects has a more facilitating effect on the students,"* whilst T2, in support of previous statements made, expressed that love of math is an important issue, saying: *"I think that mathematics plays an important role in understanding problems. So, it is easier to explain the lesson, and is especially interesting for math lovers."*

Besides the perceived advantages of interdisciplinary collaboration, the teachers stated certain disadvantages to the program as well. T1 stated seeing difficulties with the project, saying that *"Integrating into the real-life while producing the project subject requires a little more effort and creativity than the classical method,"* whilst T2 considered that such a course has a more compelling effect for those who do not like mathematics, stating that *"It was more challenging for those who didn't like math. It is important to use the right terms in the field of mathematics, and to explain problems using the right expressions."*

The teachers were also asked for their thoughts on the TA of teaching programming languages. According to the teachers, there were some advantages to lecturing according to the classical method. For example, T1 stated that *"abstract subjects can*

be embodied with real-life examples," while T2 mentioned that *"describing problems remains abstract; I don't think it addresses every student."* On talking about the perceived disadvantages of teaching programming language through the TA, T1 stated that *"Transformation of real-life scenarios into coding makes it easier for students to comprehend the subject,"* whilst T2 stated that *"It makes it easier for students to comprehend the transformation of real-life scenarios mentioned in programming."* From the interview responses of the two teachers, students may experience problems from the classical method of teaching programming in terms of transferring real-life conditions to programming processes. This situation was emphasized by the teachers as a general condition associated with the classical teaching method for programming learning.

After capturing the teachers' opinions regarding their perceived advantages and disadvantages of both the TA and IA of teaching programming, the teachers were asked which method they would prefer to use in the future. On this point, T1 stated preferring mathematics integrated into programming instruction, saying that: *"Using mathematics may be the method of choice when it comes to programming, because I think it is more effective to explain topics such as loop-variable with elements such as equations and mathematical formulae."* Although T2 held similar ideas on this issue, T2 stated preferring to return to the classical method of teaching programming in cases where the integrated teaching was not well enough understood, saying that: *"I might prefer to use math, but where* [a topic is] *not understood, I would switch back to the classical method."*

The fourth question asked to the teachers concerned any difficulties they faced during the application of the process of the current study. The teachers shared different opinions with regards to this question. T1 stated that, *"After the lecture, the projects requested from the students could be enriched."* T2 shared facing certain problems due to the general functioning of the school's schedules that had to be followed according to the educational semester, and stated that *"There were some problems due to the functioning of the school. At the end of the study, the results of the*

achievement test applied to the students were shared with them, but the results of the posttest were not shared."

The next question asked to the teachers concerned what should be included in lesson plans prepared for the teaching of programming integrated with mathematics. T1 proposed the combination of student work, gradually going from piece to piece, saying that: *"Large semester projects can be combined with tiny elements that interconnect with each other to create a whole course project plan. In this way, by the semester end, students could have produced a program covering all subjects,"* whilst T2 stated that: *"The lesson plans were very directive and I didn't feel anything more needed to be added, so I have no need to make any suggestions."*

The sixth question focused on the teachers' opinions about integrating mathematics in the teaching of programming by way of interdisciplinary collaboration. T1 stated that: *"I am very positive about the integrated use of mathematics in programming. Personally, I think it would be advantageous to use mathematics and mathematical applications in programming, both as a lifelong programming learner and as a programming instructor."* T2 stated that it would be more appropriate to include mathematical problems in programming teaching instead of directly, stating that, *"It may be more effective to integrate mathematics into the subject by asking students to make some calculations within the subject, not by giving them direct mathematical problems."*

In the final question, the teachers were asked whether they wished to share any other thoughts about the research study process. T1 expressed an appreciation for the opportunity to apply lesson plans with clear direction, whilst T2 stated that the content should be richer still, saying that: *"Instead of focusing on two areas, Classical and Math, we could keep the content richer. However, I think it was necessary to choose two fields. We could make online mid-term assessments and share their answers with students instantly. Thank you for all your efforts and I wish you success in your work."*

Teachers were asked these two questions to detect (find out) how their teaching methods affect their teaching enthusiasm: *"Which teaching method do you feel a tendency to use in your lessons; traditional method or interdisciplinary method with the integration of mathematics?" "How did your preference affect your desire to lecture?"* T1 stated that he preferred to teach with an IA because it made the abstract subject more tangible for the students. He also stated that *"at the beginning, it did not affect my teaching desire, however, after observing my students' enthusiasm and active participation in the classes where I used the interdisciplinary method, I became one of the staunch supporters of interdisciplinary collaboration in teaching programming"*. T2 indicated a similar view, *"I saw the lights of curiosity in the students' eyes when I was teaching programming with mathematics. It didn't affect my teaching passion, so I taught with the same passion to both groups, however, it made me feel more excited when I taught programming with interdisciplinary collaboration."*

4.2.3 Research Question F: Students Thoughts about Visual Programming Tool

This subsection presents the findings in response to the sixth research sub-question (f): *"What do students think about visual programming tool and its processes?"*

In order to answer this research sub-question, data obtained from interviews held with the selected participant students were analyzed through content analysis. The researcher's interviews were conducted with 24 students, representing three students from each study group. The interviewees were selected according to their academic level in mathematics. Level 1 was applied to a student with the highest GPA in their class, whilst Level 2 held a mid-level GPA, and Level 3 was applied to the student with the lowest mathematics GPA in the class. Based on this, 12 students were interviewed from the experimental groups and 12 from the control groups.

The responses of the students to questions asked to them about Small Basic teaching were analyzed according to content analysis. As a result of this process, the students' views on the teaching of interdisciplinary coding with Small Basic were evaluated separately in the context of the five questions they were asked. By creating categories related to each question, the codes obtained from the students' interview transcripts were associated with the categories.

In reporting excerpts and opinions of the interviewed students, their mathematics academic achievement level is shown in brackets, along with the code assigned to each interviewed student by the researcher. Students were assigned a code with an "E" to denote being from an experimental group, or "C" if they were from a control group. Similarly, the code "L1" denotes a student at Level 1 (highest GPA), whereas "L2" means a student at in Level 2 (mid-level GPA), and "L3" assigned to a student at Level 3 (lowest GPA). For example, SE3 (L3) means Student 3 was from an experimental group, and had the lowest math GPA in their class (hence, Level 3).

The first two questions asked to the students in the interviews related to their personal characteristics. The third question asked to the students concerned whether or not they had used programming languages or tools prior to the commencement of the study.

The third interview question posed to the students concerned the programming tools they had used prior to the study. While five students from the experimental groups had not previously used any type of coding language before, seven of the students had done so. It was noted that the most successful students from the experimental groups (bar one) stated having previously used coding languages prior to the study. From the control group, four students stated not having previously used coding languages or tools before, whilst eight of them stated having some previous experience with different programming languages. Of the four control group students with no previous programming experience, three were of low academic achievement, whilst the other was of moderate achievement, according to their GPA.

The programming languages previously used by the experimental group students mostly related to the use of the JavaScript, Kodu Game Lab, Scratch, Small Basic, Mindstorm, Arduino, mBlock, and O-Bot programming languages and tools. Examples of statements taken during the experimental groups' students include the following: SE9 (L3): *"I've done programming using Small Basic before, and I've also used the Kodu Game Lab and Scratch." and* SE7 (L1): *"Yes; Mindstorm, Scratch, Arduino Cscript, and Mbot."*

When the programming languages and tools used by the control groups' students were examined, the most commonly used prior to the study were JavaScript, HTML5, mBlock, Scratch, Small Basic, C ++, Arduino C, Python, C #, Visual Basic, and Code.org. Example statements taken from the interview transcripts of the control groups' students include the following:

- SC2 (L2): *"HTML5 and JavaScript."*
- SC7 (L1): *"Yes, I used it. I worked with Arduino C and Scratch. We were also taught Small Basic related lessons in the software course."*
- SC10 (L1): *"Yes I used. I know Python, C#, Visual Basic, and Arduino."*

The next interview question directed to the students was to gauge their positive and negative opinions about the Small Basic programming language.

The opinions obtained from the experimental group show that 10 out of the 12 students interviewed mentioned various negative aspects such as features missing from the program, whilst two students stated having no negative opinion with regards to the Small Basic programming language. In considering the control groups' students' opinions, eight students mentioned Small Basic having certain negative features, whilst the other four students indicated that the programming language was easy to learn, but that it was not considered to be very functional.

Table 27. presents the codes assigned for the negative opinions of the experimental and control groups' students about the Small Basic program, which are classified

according to the three aforementioned levels of academic achievement in mathematics.

Table 27. Codes Created for Students' Negative Opinions about Small Basic

Achievement level	Experimental group	Control group
Level 1	Limited content	Boring
	Lack of Turkish support	Very easy
Level 2	Non-consideration of codes	No complex function
	Lack of preview of codes	Limited text color
		Some items required hard work
Level 3	Too many numbers / too much text	Difficult to understand
	Difficult problems require a lot of thinking	Complex interface
	Doubts about usefulness	Continuous change of encoding type

According to the experimental groups' students, whilst using the Small Basic program is not considered to be very difficult, its functionality is somewhat controversial. For example, the student coded as SE1 (L1) said that, *"It could have been a bit more extensive."* Student SE3 (L3) was less enthused, stating that: *"I don't think I will use it in the future."* Some of the students from the experimental groups mentioned that the program was seen as challenging, and that it required some time to achieve competency in using its functions. Notably, student SE12 (L3) remarked that *"There were examples that required me to think too much. Sometimes I couldn't do the problems I wanted to do with simple thinking."* Student SE24 (L1) stated that the program was easy to use; however, language support was needed for some of its features: *"I would like the program to have Turkish language support, as English is sometimes confusing."*

The opinions of the control groups' students about Small Basic in general was that they found it easy to use, but that they considered its functions not to be satisfactory. For example, student SC8 (L2) stated that instead of having to work hard in using the program, games could be written easier using another program: *"Small Basic is just for preparing a speech and basic graphics, I would have done so much more*

coding as a game." Student SC10 (L1) also made similar comments, saying that, *"It's a very easy application for someone who knows how to program. So, I didn't give it much attention,"* and student SC5 (L2) said that *"It does nothing but write and use simple functions."* There were also some students who saw the program as uninteresting. For example, student SC1 (L1) described the program as *"boring,"* while SC3 (L2) expressed that: *"A different type of coding was applied all the time, and some of us did not like this programming and felt it was boring."*

On the other hand, some of the students complained about difficulties in using the program. Accordingly, student SC6 (L3) could not envision using the program because it was considered difficult to understand. Similarly, student SC11 (L2) stated that *"It was a bit difficult for me. I had to ask questions to my teacher and to my friends."*

When the students were asked for any positive opinions about using the Small Basic programming language, the codes obtained from the content analysis from both the experimental and control groups' students are as presented in Table 28.

Table 28. Codes Created for Students' Positive Opinions about Small Basic

Achievement level	Experimental group	Control group
Level 1	Easy to use	
	Creates productivity	Understandable language, easy code finding
	Easy to write simple code	Good option to start coding
		Easy and intuitive interface
Level 2	Writing code that can control the computer	Easy to use and understand
	Has a good purpose	Easy to learn
	Comfortable	
Level 3	Strengthens the perception of coding	Has automatic code completion
	Simple structure	
	With ready codes	

Students from both groups generally agreed that Small Basic was good for starting to learn programming because it was not considered confusing, and was seen as easy

to learn. The experimental groups' students shared their opinions with details about their usage of the program, stating that the program could be more functional, while the control groups' students also shared their thoughts about the use of Small Basic and its features.

According to the experimental groups' students, one of the most positive features of the program is that it is easy and fun to use. Student SE1 (L1) stated that *"It's easy to use. I can write the programs I want. I think it will help me in the future,"* and student SE4 (L1) said that *"it was fun coding, I even started using my own calculator at home."* Student SE5 (L2) talked about being able to control a computer using Small Basic, and stated that: *"It was fun to write a program that I talked to, like Siri. I learned how to make the computer talk and take certain actions."* Some of the students shared their opinions about programming with Small Basic as it being capable of both simple and complex coding as it was perceived to be both easy and challenging at the same time. As an example, student SE7 (L1) said that *"Beginners can create very good complex coding, and simple code is very easy to write too."* According to student SE8 (L2), *"the program is very understandable and comfortable to use,"* while student SE12 (L3) indicated that *"the ready code that pops up made the programming easier."* Among those interviewed from the experimental groups, some students found Small Basic to be fun to use. Student SE10 (L2) said that *"It helped me to be able to create fun code. For example, it was fun to present the instalment cash sample with the program which was something we use in real life,"* and student SE11 (L2) was similarly supportive, saying *"I did it* [coding] *with fun."*

The positive aspects of Small Basic programming as stated by the control groups' students are mostly about it *"being easy, easy to understand the language, with automatically entering the first letter of the code and with its fun activities." Student* SC10 (L1) highlighted that *"It can be nice for new learners. The interface we used was easy and straightforward,"* whilst student SC4 (L1) stated that the program featured *"understandable language, easy code finding,"* and student SC5 (L2)

mentioned that *"it was easy to understand and write."* One particular positive feature of Small Basic was stated by student SC6 (L3), saying that *"Some code automatically generates when we type the first letter, and that made me code easier."* In addition, student SC8 (L2) stated that the program was very effective and basic for teaching coding, saying, *"I have learned how to program through Small Basic."* Even though most of the students held positive opinions, three students expressed having more negative perceptions of the program, with student SC12 (L3) even saying, *"I cannot see a positive side because it was difficult for me."*

The fifth interview question asked to the students sought to elicit their opinions with regards to what made learning programming through Small Basic challenging for them. The students' responses to this question are again grouped according to the same three levels of mathematic achievement based on their GPA, and are presented in Table 29. The table includes the content analysis codes that were assigned according to the experimental and control groups' students' responses regarding the challenging aspects of using the Small Basic program:

Table 29. Codes for Students' Difficulties in Using Small Basic

Achievement level	Experimental group	Control group
Level 1	Understand difference between "for loop" and "while loop."	Selecting foreground color. Most colors did not work.
	Adding and subtracting with a Reading number.	Code can be simple to write.
		Supports all HTML colors.
Level 2	Experienced difficulty in processes.	All sections are equally difficult.
	Nested-Ifs were confusing.	Transfer problems to code.
	Keep codes in mind.	Repeat sections.
	Problems that need a comparison	
Level 3	Trying to write mixed codes.	Write and keep codes in mind.
	Math questions.	Had a hard time with loops. Unsure which to use now.
		All very difficult.
		There was no easy part at all.

When the assignment of the content analysis codes were examined, notably, three students from the experimental groups did not state anything negative about using Small Basic, while the other nine experimental groups' students stated experiencing certain difficulties learning Small Basic. Student SE4 (L1) said that *"It was difficult to understand the difference between 'for loop' and 'while loop,' but the math examples made it a little easier."* Student SE5 (L2) experienced difficulties, and indicated that *"The nested things became mixed,"* whilst student SE6 (L3) stated facing challenges with Small Basic in "*trying to write complex codes."* One final compelling example was from student SE7 (L1), who warned *"Do not add and subtract with the reading number."*

When the students were asked about the challenging features of the process, student SE9 (L3) stated *"…being able to practice what I had learned, because I was a little confused and frightened."* In terms of difficulties faced in the mathematical processes, student SE12 (L1) stated one of the main reasons was a negative attitude towards the mathematics lesson, saying that: *"I had difficulty with forced math questions. I did not pay much attention because I do not like mathematics in general."*

When the same questions were asked to the students from the control groups, two of the students mentioned facing no difficulties: with one saying that there was no *"easy part"* of the program. The other nine control group students gave various examples of the compelling parts of the program. SC10 (L1) complained that not all HTML colors were supported by Small Basic, and stated that *"I did not face difficulties because I had some programming knowledge."* Student SC11 (L2) indicated that *"it is easy for me to write codes but repeated examples were difficult."* According to student SC9 (L3), all elements of the program were found to be difficult, stating that *"It was all very difficult. I only understood the writing and reading part."* The same student answered *"none"* when asked about any positive features of the program. Another issue was that the more experienced students had difficulty in writing "text window" commands, as well as feeling bored with the simplicity of the program.

Student SC1 (L1) said that: *"The commands seemed too simplified since I used the C language before. Each time writing commands like Text Window etc. was a separate problem. I was little bored because I couldn't realize physical results as with Arduino."* Student SC6 (L3) also stated thinking that the most challenging part of learning Small Basic was the loops. The most challenging feature for some of the control groups' students was in the code writing process itself, with student SC3 (L3) mentioned having to *"write and keep the codes in our mind."*

The sixth question posed to the students during their interviews was regarding what helped them the most whilst learning the Small Basic program. The opinions of the students from both the experimental and the control groups about the factors that helped them in learning Small Basic are presented in Table 30., and are grouped according to their math GPA-based academic achievement level.

Table 30. Codes Created for Factors That Helped Students in Learning to Use Small Basic

Achievement level	Experimental group	Control group
Level 1	Previous knowledge, teacher, Internet sources	Teacher
	Teacher	Codes, understandable language
	Friends, teacher	Interest in coding
	Own passion to learn programming plus math	
Level 2	Teacher	Previous programming knowledge
	Teacher, math problems	Very easy
		Teacher
		Friend, teacher
Level 3	Focusing on transaction order helped complete program process	Support from the Internet
	Each transaction and subject has supporting notes	Teacher, Internet sources
	Frequent repetition	Friend, teacher
	Solving math problems, plus math notebook helped in understand the program	

According to Table 30. most of the students in the experimental group stated that they received support the most from their teachers in fulfilling the requirements of the program, and that this increased their performance whilst learning the Small Basic programming language. Student SE1 (L1) indicated that *"My teacher was very supportive,"* which is a remarkable statement. Student SE8 (L2) also expressed how important it was to have received support from their teacher, and stated that *"The first factor was certainly my teacher. He was very understandable. It helped me to use what my math teacher taught."* Also, student SE10 (L1) supported this view, mentioning *"My background, my teacher and Internet resources."* On the other hand, student SE9 (L3) highlighted that repetition of the assigned problem helped facilitate learning Small Basic, stating that *"We repeated everything many times, and that helped me a lot to understand Small Basic."* Student SE3 (L3) expressed that *"Focusing on the algorithm carefully helped me to more easily understand Small Basic."*

In addition, student SE6 (L3) pointed out that the program contained explanatory information about its functions, and that made it easier to use the program. The student also said that *"there is information for each function and item under that item."*

The statements from the control groups' students were similar to those raised by the experimental groups' students, and emphasized the significance of the teacher factor in the teaching of Small Basic. Examples taken directly from the students' statements that it was their teacher that contributed the most in their learning of the program are as follows:

- SC1 (L1): *"My teacher; however, I still have a lot that I couldn't understand."*
- SC6 (L3): *"My teacher and Internet resources."*
- SC8 (L2): *"I got help thanks to the teacher and I learned new code with their help."*

- SC11 (L2): *"My teacher and my friends."*

Additionally, student SC9 (L3) expressed not liking programming, but having been able to receive help from both the teacher as well as from friends. Student SC9 (L1) stated not needing the help of others due to prior knowledge of programming being sufficient to understand Small Basic, and stated that *"Since I already know how to program, I didn't add anything new."* Also, two of the control groups' students stated that the simplicity of the program as having had a positive effect on learning Small Basic without a struggle. Student SC4 (L1) mentioned ease of learning that included *"codes, understandable language,"* and, according to student SC5 (L2), *"the program is very easy* [to learn]*."*

The seventh and final question put to the student interviewees was to elicit their opinions about learning programming using the Small Basic program. The views of both the experimental and control groups' students on this subject are grouped according to their math academic achievement level in Table 31.

Table 31. Codes Created for Students Learning Programming Using Small Basic

Achievement level	Experimental group	Control group
Level 1	Entertaining	Boring
	Simple codes	Amusing
	Superficial activities	Learning is not entertaining
		Simple
Level 2	If everyone learns, programming can be used everywhere	Slow learning process
	Makes people think	Boring
	Has potential	Teaches the basics of programming
	Duration may be longer.	Difficult activities
Level 3	Tutorial aspect is strong	Examples are difficult and not always understandable

According the data collected from the students' interviews, on the seventh question, four students from the experimental groups did not put forth any suggestions about

the process, whilst the remaining eight students stated that learning programming with Small Basic was efficient; plus some students mentioned a few suggestions. Therefore, the verbal findings show that students in the experimental group held differing opinions regarding learning programming using Small Basic. The following statements are from the interview transcripts:

- SE2 (L2): *"Programming can be used in any area if everyone learns how to code and understand the concept of it."*
- SE4 (L1): *"It was fun, but we also had challenging problems. The activities were entertaining and challenging."*
- SE5 (L2): *"I like Java script more, but it certainly makes more sense to use Small Basic in the solution of mathematical problems."*
- SE10 (L1): *"I would like to do more in-depth activities."*
- SE11 (L2): *"The duration of learning the program could be longer."*

Moreover, student SE8 (L2), a student from an experimental group, criticized the process because of the inability to write complex programs due to the nature of Small Basic, saying *"I would like to go deeper and write more complex programs."* Another student, SE9 (L3), stated that the program was potentially very functional, saying *"It's definitely a very instructive programming language and it makes people think."*

On the other hand, three of the students in the control group did not make any suggestions about the learning process for Small Basic, whilst three of them stated having found the program boring. While some students suggested the transition to more difficult tasks early on from the existence of programming information, others mentioned the difficulty level of the examples used in the teaching of the program. Some of the students' expressions in the control groups were as follows:

- SC1 (L1): *"It is a well-prepared program but, in my opinion, it is boring again."*

- SC7 (L1): *"Learning Small Basic was not entertaining. I would prefer the teachers to use Arduino C. However, it might be good for the new starters to learn Small Basic because it is easy to learn."*

The fact that the program does not have advanced levels was seen as a handicap by student SC10 (L1), stating *"I think there may be a different alternative for those who already know* [programming]. *But I think it was beneficial for my friends who do not have any programming language background."*

There were also some students who held opposing views on the difficulty of using the Small Basic program, such as the following:

- SC2 (L2): *"We could have moved on to more advanced things earlier."*
- SC5 (L3): *"We could work with easier and more understandable examples."*
- SC12 (L3): *"I wish there were easier activities."*

4.3 Summary of Findings

In order to find an answer to if there is any significant difference between the control and experimental groups according to their;

- Mathematics Attitude Scale (MAS) scores,
- Computational Thinking Skills Self Efficacy Scale (CTSSES) scores,
- Computer Programming Skills Self Efficacy Scale (CPSES) scores,
- retention of Small Basic Programming Achievement Test (SBPAT) results,

Paired Group t-test, independent sample t-test and ANCOVA analysis are applied.

The analysis made for scores obtained from the pretest and posttest applications of the MAS scale for both groups showed as below;

- Paired t-test showed that the experimental group's pretest of the MAS showed a mean score of M = 3.03 (SD = 0.28), while the mean score for the control group was M = 2.88 (SD = 0.24). While the students in the experimental group obtained a mean score of M = 2.87 (SD = 0.17) from the second application of the MAS, the students in the control group obtained a mean score of M = 2.83 (SD = 0.16).
- Independent sample t-test analysis stated the pretest results show that the attitudes of the students in the experimental group towards mathematics showed a significant difference ($t(186) = 3.914$; $p < .00$). In terms of the posttest results, even though the extreme values had been removed, no statistically significant difference was found to exist between the mean scores of the experimental and control groups ($t(177) = 1.635$; $p > .05$).
- Finally the ANCOVA analysis result showed that the pretest scores did not have any effect on the posttest scores ($F(1.066)$, $p > .05$).

The analysis made for scores obtained from the pretest and posttest applications of the CTSSES scale for both groups showed as below;

- Paired t-test showed that students in the experimental group achieved a mean score average of $M = 40.29$ ($SD = 13.81$) from the pretest application of the CTSSES whilst the mean score average of the students in the control group was $M = 44.57$ ($SD = 16.11$). The posttests result of the experimental group students was $M = 58.74$ ($SD = 11.87$), whilst the control group students' mean scores averaged $M = 43.31$ ($SD = 13.51$). When these findings are compared, it can clearly be seen that the students in the experimental groups achieved a higher mean score average compared to the students from the control groups.
- Independent sample t-test analysis stated that the experimental group students' CTSSES differed significantly between the pretest ($t(186) = -1.96$; $p < .05$) and posttest ($t(186) = 10.99$; $p < .01$) scale applications.

- ANCOVA analysis findings showed that the students' pretest scores did not influence their posttest scores ($F(1, 181) = .233$; $p > .05$).

The analysis made for scores obtained from the pretest and posttest applications of the CPSES scale for both groups showed as below;

- Paired t-test showed that score for the experimental groups' pretests was $M = 6.05$ ($SD = 2.38$), whilst for the control groups' pretests it was $M = 8.16$ ($SD = 2.97$). The average posttest score for the experimental groups was $M = 11.14$ ($SD = 1.71$), whilst it was $M = 8.01$ ($SD = 2.27$) for the control groups.
- Independent sample t-test analysis stated that there was a statistically significant difference found between the experimental and the control groups in terms of their pretest scores ($t(186) = -5.39$; $p < .01$) and posttest scores ($t(180) = 12.98$; $p < .01$).
- ANCOVA analysis findings showed that the pretest scores did not have a statistically significant effect on the posttest ($F(1, 179) = 152.58$; $p > .05$).

The analysis made for scores obtained from the pretest and posttest applications of the SBPAT scale for both groups showed as below;

- Paired t-test showed that the students in the experimental group obtained an average mean score of M = 83.32 from the pretest application of the Small Basic Programming Achievement Test, whilst the control groups' students obtained an average mean score of M = 67.98. The average mean scores from the posttests were M = 84.28 from the experimental groups' students, and M = 60.32 for students of the control groups. The mean scores achieved by the experimental groups' students increased slightly by +0.96 points in the posttest. However, for the control groups' students, their posttest Small Basic Programming Achievement Test results were much lower, representing a decrease of -7.66.

- Independent sample t-test analysis stated that there was a statistically significant difference found between the experimental and the control groups in terms of their pretest scores and posttest scores.
- ANCOVA analysis findings showed there was a difference observed between the average scores of the experimental and control groups pretest and posttest scores for the SBPAT, and that the difference was shown to be statistically significant (Wilks Lambda = .915, $F = 19.669$, $p < .001$).

Qualitative findings for the teachers showed that both T1 and T2 have similar teaching experience with programming languages. T1 stated having had no difficulty in teaching programming languages to the students, whilst T2 sometimes stated sometimes having had difficulty finding the right subjects. Both teachers agreed on having mathematics as second discipline in programming course had the effect of increasing students' motivation to learn. They agreed on it is an advantage for students who have tendency on mathematics, but it became a disadvantage for students who do not like mathematics at all. When teachers asked if they'd like to continue teaching programming with interdisciplinary collaboration, both are agreed and said "yes". However, they also added, students can be offered with more disciplines instead of just having mathematics.

Qualitative findings for the students showed that students in both groups have the similar programming background according to their level of GPA scores. Students with high Mathematics GPA level (L1) have more experienced than students with low Mathematics GPA level (L3). the experimental groups' students saw the Small Basic program as having functionality that was somewhat controversial, although its usage was not considered to be difficult. Similarly, the students from the control groups stated that Small Basic was an easy program to learn and use, but that its features were inadequate and therefore of limited functional use. When students were asked the challenging part of the process, experimental group students stated mathematic questions as control group students stated only the using of programming terms. When the interviewed students were asked about which elements helped them

the most whilst learning programming with Small Basic both groups stated the similar things. However, the experimental groups' students stated mathematics course book differently than the control group students.

CHAPTER 5

DISCUSSION AND CONCLUSION

This chapter provides a discussion for each sub-research question in the study, relating the findings of the current study back to the published literature. Therefore, the results obtained in the current study are discussed in this chapter under six subsections, with relevant examples provided from the literature related to each of the six research sub-questions. This is followed by the study's conclusion, after which the researcher suggests the implications of the study and outlines possible future research.

5.1 Discussion

5.1.1 Teaching programming approaches and Mathematics Attitude

In order to determine if a difference existed between the control and experimental groups, the students' attitude towards mathematics was measured using the Mathematic Attitude Scale, as developed by Duatepe and Çilesiz (1999), which was applied both as a pretest and also as a posttest during the study.

First, the mean average scores that the students obtained from completion of the scale were examined. The H_0 is rejected because findings showed that the mean scores obtained from the posttest for both the experimental and control groups decreased when compared to the pretest results.

When the related studies were examined in the literature, some supported the finding that information technologies and the integrating of computer science into

mathematics have a positive effect on students' attitudes towards mathematics (Aktümen & Kaçar, 2008; Fábián et al., 2008; Kramarski & Zeichner, 2001; Rodríguez-Martínez et al., 2020). For example, in a study conducted by Aktümen and Kaçar (2008), which examined students' attitudes towards mathematics after the integrating of Maple computer algebra software into mathematics lessons, the study's findings indicated that teaching using the computer program positively affected the attitudes of the students towards mathematics. Similarly, in a study conducted by Fábián et al. (2008), it was observed that teaching mathematics with the help of mobile teaching technologies contributed to the students' positive attitude, as well as supporting the students' participation in the lesson and their academic success. In many studies, it was observed that applications supported by information technologies develop positive attitudes in students towards mathematics courses, increase their level of thinking and reasoning, support conceptual learning, and contribute to the learning of mathematics in an effective and enjoyable way (Fırat, 2011; Gürbüz & Birgin, 2012; Kebritchi et al., 2010; Kramarski & Zeichner, 2001; Polaki, 2002; Pratt, 2000).

However, the results of the current study for this first research sub-question showed an opposing view when compared to the aforementioned studies. Therefore, the extreme values were deleted from the dataset and the measurements retaken again. However, similar results to the first attempt were also obtained on the second. Even though a difference was observed between both of the experimental and control groups in their pretests and posttests, the results were not found to be statistically significant in their difference when the posttests were compared back to the pretests. In addition, no significant effect was observed in the covariance analysis performed in order to determine whether or not the pretests influenced the groups' scores in the posttest. However, this result was an unexpected situation for the researcher. In the Sobel analysis conducted to examine the details of this finding, it was observed that the Mathematics Attitude Scale pretest results had a positive effect on the posttest, although this was not found to be statistically significant. Therefore, the literature

was reexamined to look for the reasoning for similar negative effects seen for the use of information technology on students' mathematics attitude.

When the related literature was analyzed, it was seen that many factors can affect students' attitudes towards mathematics lessons. Hannula (2002) proposed a new conceptual framework for analyzing attitude and changes in attitude. The author discussed students' attitudes towards mathematics by dividing them into four evaluation processes: 1) emotions experienced by students during activities related to mathematics; 2) emotions that students automatically associate with mathematics concepts; 3) evaluations of situations that students expect to follow as a result of doing mathematics; 4) the value of mathematical goals in students' global structure.

In the current study, when the reasons for decreased students' attitudes towards mathematics were examined, the attitude of teachers who participated in the study were also included. Based on the participant teachers' responses, within group and between group comparisons were made and it was observed that the effect of teachers on the mathematics attitude of the groups was limited. Contrary to the findings of the current study, some studies in the literature found that teachers' approach towards students led to the development of a positive attitude towards mathematics. In a study conducted by Mata et al. (2012), the participant teacher was observed to be successful in changing students' attitudes, beliefs, and behaviors towards mathematics. Some other research studies (Aktümen & Kaçar, 2008; Mata et al., 2012; Özgen & Pesen, 2008) have shown that attitudes can sometimes change significantly within a relatively short timeframe.

When the reasons were examined in the case of the current study, the interviews conducted with both the participant teachers and students were reanalyzed. Student SE12 (L3) from the experimental group stated that the most challenging part of the process was that the *"Math questions were really challenging. I couldn't pay enough attention to learning the program because I literally hate math lessons."* The same student also mentioned that having mathematics questions within an ICT course was

like having an additional hour of math lessons each week, which negatively affected the students' motivation.

On teaching programming with mathematics, Teacher 2 said that, *"Mathematics is good in helping to make the problems more abstract; however, it may be disadvantageous for students who do not like mathematics as a course."* Therefore, the reason why the students' mathematics attitude did not change to a statistically significant level may be due to their facing too great a level of mathematics problems, which were taken from the mathematics course book, in the programming course. It could be said, therefore, that this made the students feel as if they were having an additional mathematics course, which decreased their attitude slightly towards the course.

5.1.2 Computational Thinking Skills Self-Efficacy

In order to examine the effect of integrating math as a second discipline in the teaching of programming on students' computational thinking skills, the Computational Thinking Skills Self-Efficacy Scale developed by Gülbahar et al. (2019) was applied, both as a pretest and a posttest, and to both students from the experimental control groups.

Lye and Koh (2014) stated that the inclusion of computational thinking in computer science is a manifest part of the equation, and that programming is a less complex element than computational thinking, which requires thinking skills based on more fundamental concepts. Therefore, the literature supports that integrating computational thinking skills into programming teaching provides can positively impact on the conceptual learning of both programming and computational thinking (Hickmott et al., 2018; Lye & Koh, 2014; Rodríguez-Martínez et al., 2020). The results from the current study accepting the H_0 because they showed that the experimental groups' students were seen to have obtained a higher average from the Computational Thinking Skills Self-Efficacy scale at the end of the experimental

process (posttest) when compared to the pre-experimental process (pretest). However, the control groups' students showed a slightly lower average in the posttest application of the same scale than in the pretest application. The difference between the posttest and pretest applications demonstrates that students from the experimental groups showed a considerable increase in their computational thinking skills, whereas the control groups showed a slight decrease. Scholars appear to agree on the positive influence of programming teaching on computational thinking skills due to its high beneficially for problem-solving skills (Benitti & Spolaôr, 2017; Benton et al., 2017; Calao et al., 2015; Rodríguez-Martínez et al., 2020) which is also indicated in the results of the current study. Extreme values of the obtained scores from both groups' posttests were examined and subsequently eliminated from the dataset prior to conducting a between group comparison. The mean score of the experimental groups' students was found to be lower than those of the control groups prior to the application of the study's procedure. However, when the posttest was applied to both groups at the end of the study, it was seen that the experimental groups' students showed a higher level of performance in terms of their computational thinking. Based on these findings, it may be said that the teaching of programming according to an IA increases students' self-efficacy in computational thinking skills. These results are also supported by other researchers who have indicated that programming naturally improves computational thinking skills due to the existence of common components in algorithmic thinking, problem solving, and critical thinking (Berland & Lee, 2011; Gülbahar et al., 2019; Kılıç et al., 2020; Tsai et al., 2019).

In order to support the findings of the second research sub-question, the literature related to computational thinking skills and academic success was then examined. In the published research, qualitative studies especially can be found that examined the opinions of both students and educators. Interviews, observation, and evaluation were generally applied in the examination of participants' opinions. The studies found were mostly applied at the secondary school level (e.g., Barak & Assal, 2018; Benitti & Spolaôr, 2017; Francis & Davis, 2018; Göksoy & Yılmaz, 2018; Kasalak, 2017; Kazez & Genç, 2016a, 2016b; Takacs et al., 2016; Yolcu, 2018). Scholars

appear to agree that in case studies, information-assisted teaching practices strengthen students' computational thinking skills, and they indicate that programming activities can positively change the attitudes of students towards a second discipline, whilst also strengthening their computational thinking skills self-efficacy (Benitti & Spolaôr, 2017; Grover & Pea, 2013; Iyer, 2019; Kasalak, 2017; Sáez-López et al., 2016; Takacs et al., 2016). Moreover, many researchers put forward evaluations and evidence that students who solve problems through programming improve their computational thinking skills (e.g., Atmatzidou & Demetriadis., 2016; Bilge Kunduz, 2018; Corradini et al., 2017; Djambong & Freiman, 2016; Vieira et al., 2016). According to some researchers, complex cognitive skills such as procedural and conditional reasoning, as well as planning and attribution are employed as part of the programming process, and that they play a significant role in the development of computational thinking skills (Barut et al., 2016; Çelik & Özdener, 2019; Çetinkaya, 2019; Jun et al., 2017; Moons & De Backer, 2013; Rodriguez et al., 2017).

To add further support to the current study's findings, another subject that was examined was whether or not the scores obtained by the students from the experimental and control groups in terms of the Computational Thinking Skills Self-Efficacy Scale were affected by the two teachers who taught them during the study. Based on the two participant teachers, within group and between group comparisons were made, and it was observed that there was no effect of the teachers found according to the computational thinking skills self-efficacy of the student groups. The literature was then examined with regards to the effect and role of teachers on students' computational thinking skills, and how they integrated computational thinking into their lessons. In a study conducted by Rich and Yadav (2019), teachers used three ways to increase their students' computational thinking skills; using computational thinking to guide their own planning and thinking, using computational thinking to structure their lessons, especially for science and math, and presenting computational thinking as a general problem-solving strategy. In another study, Belanger et al. (2018) found that instructional activities in

programming courses had a strong influence on students' ability to solve problems directly related to computational thinking. In a study by Daily et al. (2015), the researchers focused on embodying computational thinking skills through the use of technological tools in the classroom, and obtained similar findings to that of the current study.

Interviews conducted with the two participant teachers in the current study also showed that learning programming can increase students' computational thinking skills. On this, Teacher 1 expressed that; *"I am very positive about teaching programming with mathematics, because it increases students problem-solving skills and their lifelong learning skills, so I think programming with the integration of mathematics is an advantage for students to develop more than one skills area."* Since problem-solving skills form an element of computational thinking, their development has a natural positive effect on the development of computational thinking skills. In another study, Fields et al. (2019) observed that teachers helped to improve their students' computational thinking skills through computer-assisted instruction and programming. Similarly, Jun et al. (2017) found that an experimental process involving students using coding programs showed how design-based learning improved computational thinking in elementary school students.

5.1.3 Computer Programming Self-Efficacy

Researchers have stated that programming should be learned during early ages, and that every single person should be aware of the importance of programming (Duncan et al., 2014; Fojtik, 2014; Guzdial & DiSalvo, 2013; Harvie et al., 2018; Oliveira Aureliano, 2013). As Kalelioğlu and Gülbahar (2014) indicated, learning programming improves the abilities of higher-order thinking, critical thinking, creative thinking, and problem solving. However, teaching programming to young learners is not considered to be as easy as teaching it to adults. Thus, Burns et al. (2012) pointed out that teaching programming through an IA facilitates learning, and that having a second discipline makes the transference of knowledge to learners as

being different from that of traditional teaching. The current study aimed to establish if and how interdisciplinary collaboration can influence the learning of programming skills. In order to determine whether or not the experimental process affected the programming skills of the participant students, the mean score averages that the experimental and control groups' students obtained from the pretest and posttest applications of the Computer Programming Self-Efficacy Scale were examined. The average posttest mean score obtained from the experimental groups increased when compared to the pretest, while the average posttest mean score was found to have decreased for the control groups' students. Therefore, the H_0 is accepted. In addition, the difference between the posttest and the pretest scale applications for the experimental groups were found to be higher than the scores of the control groups.

The extreme values of the scores obtained by both sets of student groups in the Computational Thinking Skills Self-Efficacy Scale were examined, and then eliminated from the dataset prior to conducting the between group comparison. The differentiation test results showed that the average mean score of the experimental groups' students was lower than for the control groups' students prior to the commencement of the study. However, at the end of the study period, the posttest results showed that the experimental groups' students obtained higher mean scores following the experimental process application. Co-variance analysis was then performed in order to determine if the pretest scores had an effect on the posttest scores for both groups, and no significant effect was observed. Cheng (1972) indicated that both mathematics and computer science somehow interacts with each other and with other disciplines, therefore, based on these findings, it can clearly be stated that the teaching of programming through an interdisciplinary approach can increase students' programming skills. Also, scholars appear to agree that an understanding of mathematics helps to make the understanding of programming deeper and easier to achieve, due to there being a significant connection between mathematics and computer science, and that this connection facilitates learning as both disciplines are aimed at finding solutions to given problems (Burns et al., 2012;

Fisler et al., 2020; Graham & Fennell, 2001; Lu & Fletcher, 2009; Pruski & Friedman, 2014).

In a study conducted by Jehlička (2010), research focused on the interconnection between physics, informatics, and mathematics through a computer sciences course. The findings of the study revealed that teaching programming through an interdisciplinary approach increased the participant students' motivation towards computer science, and simultaneously attracted their attention to learning and to independently join in with computer science activities. Davenport et al. (2014) conducted a research study on the teaching of programming to mathematics student on a descriptive mathematics course. Their study's findings indicated that teaching programming to math students through the integration of descriptive mathematics helped students to move away from the question *"why do we have to learn this?"* to one of *"how we can do this?"* – meaning that they started to think more in abstract terms due to working on problems based on real life.

Another subject examined in the current study was whether or not the scores obtained by the experimental and control groups' students from the Computer Programming Self-Efficacy Scale were affected by the teachers who taught them during the study. Based on the participant teachers, both within group and between group comparisons were performed, and it was observed that there was an effect of the teachers on the programming skills of the student groups. In other words, it may be stated that teacher attitude can be an affective factor in students' learning of programming skills.

When the qualitative data of the current study were examined, it could be seen that students from the experimental groups showed a more positive attitude towards learning programming than the control groups' students. For example, student SE1 (L1) said that it was fun to learn programming, and that having real-life examples, especially those from the math book, helped in being able to understand the concept more easily. Student SE8 (L2) expressed that *"It was helpful to use the knowledge that my math teacher taught to understand how to program. It made me realize I can do a lot with programming now."* On the other hand, students in the control group

felt that it was not easy to understand programming through the problems given. Student SC6 (L3) stated that *"I couldn't really understand Small Basic because it was not easy for me to visualize the problems and their solutions in my mind*," and student SC8 (L2) indicated that *"it was hard to program the given problems."*

It could be said that, teaching programming through an interdisciplinary approach helps students to visualize problems more easily. The current study has shown that programming skills develop more when taught alongside a second discipline, because it affords students the opportunity to consider abstract thinking of a problem, and provides a connection to the real world more naturally since what is used in the programming course is linked to another course they have previously taken.

5.1.4 Small Basic Programming Achievement

The purpose of Small Basic Programming Achievement Test was to measure the participant students' level of knowledge of the Small Basic programming language, which is a visual programming tool that they had been taught during the application in the current study. In order to determine whether or not the experimental process had an effect on the students' permanent retention of knowledge, the Small Basic Programming Achievement Test was applied as a pretest at the end of the application, and then repeated 10 weeks after as a posttest. The mean score averages that the experimental and control groups' students obtained from the two applications of the test were then examined. When the results were compared, the pretest and posttest average mean scores of the students in the four experimental groups were all found to be close, whilst the posttest averages of the students in the four control groups had decreased significantly from the pretest. The H_0 is accepted because the results show that the decrease observed in the posttest results of the control groups was greater than the scores seen in the experimental groups.

When the analysis findings were examined, the difference observed between the mean score averages of the experimental groups and the control groups from the

pretest and posttest results were found to be statistically significant. In addition, in the comparison of differences between the experimental and control groups, it was observed that the students' programming achievement differed significantly in terms of their pretest and posttest scores. Based on these findings, it may be said that the teaching of programming to young learners through an interdisciplinary approach can increase students' programming achievement and its retention. Denning et al. (2017) stated that solving mathematical problems is similar to solving problems using a programming language, as both require a level of logic, which is why mathematics and computer science interact with each other so well. Also, scholars have pointed out that a many people choosing careers that are related to computer science are also working in mathematics, because the relations between computer algorithms are all about mathematics (Burns et al., 2012; Knuth, 1974; Mahadeo et al., 2020; Swacha & Baszuro, 2013).

On the one hand, these processes are based on largely mathematical data; however, computers can complete these tasks or resolve problems through the application of mathematical calculations. These processes are conducted by computers at the logical level, hence, conscious intelligence or knowledge is needed in order to guide or guarantee the accuracy of the output data (Denning et al., 2017). It is for this reason that linking computer science and mathematics is deemed significant at an early stage in the K-12 process, so as to prepare students who can use and interact with computers at both the logical and conscious levels.

Another subject examined was whether or not the scores obtained by the experimental and control groups' students from the Small Basic Programming Achievement Test were affected by the two teachers who participated in the study. Based on the participant teachers, within group and between group comparisons were performed and it was observed that there was an effect of the teachers on the Small Basic Programming Achievement Test of the control groups. However, it was observed that the teachers did not similarly affect the experimental groups' students. In addition, a significant effect was observed in the covariance analysis performed

to determine whether or not the pretest affected the posttest scores of the groups according to the teacher that taught them. Based on the results of the analysis, it may be said that the teachers participating in the research had limited effect on the students' Small Basic Programming Achievement Test.

When the literature was examined to support the current study's findings, it was seen that teaching programming to young learners is considered to be challenging, because younger students learn better when there are abstract examples, which means providing a real-world connection (Bornat & Dehnadi, 2008; Grover & Pea, 2013; Jenkins, 2002; Kalelioğlu, 2015; Özmen & Altun, 2014). Mathematics was used in the case of the current study in order to provide real-world problems to students, and for combining these problems with their acquired programming knowledge so as to understand the nature of computer science more easily. Burns et al. (2012) stated that computer science can be seen as easier when it is learned alongside a second subject, because it helps to connect what is real with computers. The current study was found to be in agreement with other scholars through its findings, as the experimental student groups showed higher levels of improvement in their programming skills based on the pretest and posttest results.

The effects of interdisciplinary collaboration with regards to the retention of programming achievement was also a focus of the current study, and the results also found to support the literature. Most researchers agree that new methods of teaching methodology have been studied for almost all disciplines in order to facilitate learning and to make the knowledge gained more permanent (De Jesus Gomes et al., 2015; Guzdial & DiSalvo, 2013; Kelleher & Pausch, 2007; Maloney et al., 2008; Oliveira Aureliano, 2013; Resnick et al., 2009; Saeli et al., 2011). This need has increased further in programming learning because of its perceived level of difficulty and its inherent requirement for advanced skills such as problem solving, creative thinking, algorithmic thinking, and systematic thinking (Draganoiu et al., 2017). The involvement of a second discipline in computer science courses can establish a natural link between the real world and the machine world, with learners tasked with

thinking not only about the structure and syntaxes involved in programming, but also in solving real-world problems as well (Marciuc & Miron, 2017). The relation between computer science and mathematics is undeniable, with both focused on the solving of problems, the use of algorithms, and finding solutions for given problems (Burns et al., 2012; Celedón-Pattichis et al., 2013). Therefore, a need had already been established to examine the effect of mathematics on programming learning. The current study aimed to find out whether or not teaching a programming language with the collaboration of mathematics through an interdisciplinary approach affected the learners' programming skills, and also how it affected their learning retention. The results of this experimental research with a sample of 188 seventh-grade secondary school students showed that those who were taught programming through an interdisciplinary approach by using mathematics showed a significant increase in their computer programming skills.

5.1.5 Teachers' Thoughts on Teaching Small Basic

Interviews made with two teachers who participated as instructor to the study in order to find out their thought about the study and how interdisciplinary approach in teaching programming was different from the TA. Both of the participant teachers were interviewed, and both had a generally positive opinion with regards to teaching programming by using the Small Basic program through a mathematics course. According to the verbal data collected during the interviews, the teachers stated that performing project-based studies in programming teaching increased the effect of the teaching process, and that they believed it would increase the motivation of the students also. On the other hand, the teachers highlight that in order to achieve retention in programming learning, real-life examples should be used so that the mathematics problems may better help students to retain their new knowledge in their long-term memory.

According to the literature, teaching programming to young learners should be undertaken using basic tools such as Scratch and/or Small Basic because they have

been shown as contributory to achieving more successful results (Bishop-Clark et al., 2007; Calder, 2010; Cooper et al., 2003; Klassen, 2006; Moreno et al., 2004; Peppler & Kafai, 2007; Rajala et al., 2008). The current study's interview data also showed that using Small Basic was a good choice for teaching text-based programming to young learners. When the teachers' interview transcripts were examined, the following codes were identified as key from the content analysis:

- problem solving;
- computational thinking;
- algorithmic thinking;
- mathematic success and programming success are directly proportional; and,
- real-life samples.

In a study conducted by Akçay (2009), both the advantages and disadvantages of teaching Small Basic to young learners were examined. During the study, Akçay sought the opinions of participant teachers, and it was observed that using the Small Basic program provided increased levels of motivation to the students during the instructional process. Also, many similar studies concluded that Small Basic and similar programs can be affective on students' programming learning. When the opinions of the teachers and students were examined, it was observed that such programs made the following contributions to the teaching process and also to the students' learning (Bishop-Clark et al., 2007; Calder, 2010; Cooper et al., 2003; Fesakis & Serafeim, 2009; Kaucic & Asic, 2011; Klassen, 2006; Moreno et al., 2004; Peppler & Kafai, 2007; Prawalpatagool, 2010, as cited by Uzun & Baltalı, 2020; Rajala et al., 2008):

- better understanding of abstract concepts;
- increased academic success;
- easier understanding of programming languages;

- increased student motivation;
- strengthening of programming skills;
- students more effective during the teaching process;
- strengthening of algorithmic thinking;
- basic codes to increase retention;
- strengthening of students' self-confidence;
- supports better understanding of math concepts; and,
- strengthening of mathematical thinking.

Both of the teachers who participated in the current study agreed that teaching programming through an interdisciplinary approach increased not only their students' programming skills, but also their algorithmic thinking, computational thinking, and problem-solving skills. On this issue, Teacher 1 stated that, *"Using mathematics as a second discipline in programming teaching was useful because it helped when I taught students about loops, which is related to mathematical formulae and problems."* Similarly, Teacher 2 said that the *"Interdisciplinary approach was helpful; however, the second discipline could also be enriched with other disciplines too, so as to help students in the learning of programming more for those who have less interest in mathematics."* As both teachers stated, integrating other disciplines into the teaching of programming can facilitate programming learning for young learners. The participant teachers commented positively on the teaching method they chose by emphasizing the students' excitement when they were learning programming with mathematics. The teachers agreed that it was an exciting experience for them, too. Even though, both teachers said they taught fairly to both groups, observing success in teaching programming with interdisciplinary collaboration might affect their passion for teaching which may affect students' learning similarly.

5.1.6 Students Thoughts about Small Basic Programming

In order to find out students perception about the process of the study, an interview was conducted with 24 students selected from the 188 who participated in the study, with 12 students representing the experimental groups and 12 from the control groups. The students were chosen from each of the groups according to their mathematics GPA level, based on three levels according to the highest, middle-ranking, and lowest GPA scores within each respective class group. Five predefined questions were asked to the interviewed students. Content analysis was then conducted on the interview data, with a variety of different and similar codes created according to their academic success levels and which student group they belonged to (experimental or control).

According to the verbal data, five of the interviewed students from the experimental groups stated their having not previously used a programming language prior to the study, whilst seven of the students stated that they had previously studied one or various programming languages. Students with the highest mathematics GPA scores in their respective classes from the experimental groups (except for one) had previously used at least one programming language. As to the students from the control group, four stated having not previously used a programming language or tool prior to the study, whilst eight students indicated that they had. Three of the control groups' students with no previous programming experience were ranked as having low academic achievement in mathematics, whilst one was ranked as having moderate achievement. In both groups, where applicable, similar programming languages and tools had been used prior to the study. However, in addition, the control groups' students differed from the experimental groups' students in having also used programming languages such as HTML5, C ++, C #, Python, and Visual Basic.

When the interview documents of the experimental group students were examined, 10 of the students mentioned various negative opinions with regards to the additional

features they wanted from the Small Basic program, with the other two students mentioning no negative aspects regarding the program. Eight of the students from the control groups mentioned certain negative features of the Small Basic program, whilst the other four found the program to be quite simplistic and therefore considered not very functional.

When asked about their view on the use of Small Basic, the experimental groups' students saw the Small Basic program as having functionality that was somewhat controversial, although its usage was not considered to be difficult. Similarly, the students from the control groups stated that Small Basic was an easy program to learn and use, but that its features were inadequate and therefore of limited functional use. According to the content analysis codes obtained from the experimental groups' students, the negative features of the Small Basic program included: *"content is limited"*; *"there is no Turkish* [language] *support"*; *"no previews of codes"*; *"codes are not easily visible"*; *"the program contains a lot of numbers and text"*; *"it contains difficult problems that require a lot of thinking"*; and, *"there are doubts about its usefulness."* When the same question was asked to the students from the control groups, the following codes were obtained: *"boring"*; *"very simple"*; *"no complex functionality"*; *"limited font colors"*; *"very little work"*; *"hard to understand"*; *"complex interface"*; and, *"constant change of coding type."*

Upon asking the students about the positive aspects of the Small Basic programing tool, the experimental groups' students stated that it was better to use the program when starting out with learning coding because it facilitated learning programming. On the other hand, the control groups' students stated that the Small Basic program could be useful in more general terms due to the positive features of the tool. Different to the control groups, the experimental groups' students provided detailed explanations on which aspects of the Small Basic program could be more functional, which types of programming could be undertaken with the tool, and how it could form the basis for the learning of subsequent programming processes.

When the students' positive opinions about the Small Basic program were examined, it was seen that the experimental groups formed the following codes: *"easy to use"*; *"provides productivity"*; *"easy to write simple codes"*; *"print codes that can control the computer"*; *"has a good purpose"*; *"comfortable"*; *"strengthens the perception of coding"*; and *"has a simple structure and ready codes."* According to control group students, the positive aspects of the Small Basic program were seen as: *"understandable language"*; *"easy to program"*; *"good option to start coding"*; *"an easy and understandable interface"*; *"easy to understand and write"*; *"easy to learn";* and *"has automatic code-tag completion feature."*

The interviewed students were asked about what they saw as the most challenging part of using the Small Basic program. Three of the experimental groups' students stated that no part of the Small Basic program challenged them, whilst nine of them provided details about what they considered to be the more challenging aspects of the program. Two of the control group students stated experiencing no difficulties, one mentioned there being no *"easy part"* of the Small Basic program, whilst the other nine students gave various examples. The content analysis codes created for the experimental groups' students with regards to the challenges of using the Small Basic programming tool were as follows: *"To understand the difference between 'for loop' and 'while loop'"*; *"adding and subtracting with reading number"*; *"difficulties in processes"*; *"the 'nested ifs' were confusing"*; *"remembering the codes"*; *"problems that needed comparison"*; *"trying to write mixed codes"*; and *"understanding the math questions."* The content analysis codes created for the control groups' students were as follows: *"choosing a foreground color, as most colors did not work"*; *"to support all HTML colors"*; *"all sections were equally difficult"*; *"transferring problems to code"*; *"repeating sections"*; *"writing and remembering codes"*; *"having a hard time with 'loops'"*; *"not knowing which codes to use again"*; *"all aspects were very difficult"*; and *"there was no easy part."*

When the interviewed students were asked about which elements helped them the most whilst learning programming with Small Basic, the experimental groups'

students stated the following: *"my previous programming experience"*; *"my teacher and Internet sources"*; *"my teacher"*; *"my friends and my teacher"*; *"my passion about programming and math"*; *"my teacher and math problems"*; *"focusing on the transaction order helped me finish the program"*; *"each transaction and subject had supporting notes"*; *"frequent repetitions"*; and *"problem solutions in the math notebook helped."* The content analysis codes for the control groups' students for the same question were as follows: *"teacher"*; *"codes"*; *"understandable language"*; *"my interest in coding"*; *"having coding knowledge made it very easy"*; *"my friend and my teacher"*; *"having support from the Internet"*; and *"my teacher and Internet sources."*

The experimental and the control groups' students held both similar and differing opinions with regards to learning programming using the Small Basic programming tool. While four of the experimental groups' students made no suggestions, eight of them stated that programming instruction with Small Basic was efficient and some suggestions were also provided. The findings on this subject showed that the experimental groups' students held differing opinions about the use of Small Basic in the teaching and learning of programming. Three of the students from the control groups made no suggestions about this, whilst three stated that they found the Small Basic program to be boring. Even though some of the students suggested that the transition to the next level should be include more difficult programs, others mentioned that the difficulty of the examples used in the teaching of the Small Basic program caused them to be slow in learning programming.

The experimental groups' students explained about their experiences in using the Small Basic program with the following statements: *"fun"*; *"codes are simple"*; *"superficial activities"*; *"programming can be used everywhere if everyone learns"*; *"makes people think"*; *"has potential for creative thinking skills"*; *"can be taught for longer"*; and *"has a strong instructive direction."* The control groups' students' codes regarding their experience with using the Small Basic program were as follows: "*boring"*; *"fun"*; *"not enjoyable to learn"*; *"simple"*; *"slow learning*

process"; *"teaches the basics of programming"*; *"activities difficult"*; *"examples not difficult"*; and *"understandable."*

When the related literature were analyzed, it was seen that students were generally positive about programming teaching, but also expressed certain negative opinions arising from the nature of the instructional process. In a study conducted by Güleryüz (2019), it was observed that the participant students were able to prepare games using their computer and also produced auto-functioning codes (robots) after having received sufficient programming training. In the study conducted by Çetin and Günay (2011), it was concluded that students were happy to see educational topics such as experimenting, playing games, and preparing animation during their coding education. These can be considered in some way as expected results. Similar findings were also found in a study by Miceli et al. (2013), where the participating students were happy to use programming in education, and mentioned the effective functions that could be used across numerous disciplines.

In many of the published studies, it was observed that the participant students had various ideas about programming, and a number of suggestions about new products that they wished to develop. Also, that, as their knowledge about programming increased, their thinking diversity with regards to their discipline increased. In these studies, the students generally expressed opinions that through the instructional activities carried out in various programs, their problem-solving skills had improved, and they could view the events from different perspectives, and thereby their imagination had improved, their psychomotor skills had strengthened, and their desire to produce a product had increased (Gerecke & Wagner, 2007; Göksoy & Yılmaz, 2018; Lin et al., 2009; Liu et al., 2013).

As a result, it could be said that programming instruction conducted using the mathematics discipline and the Small Basic program through an interdisciplinary approach does not greatly affect the students' attitudes towards mathematics, whilst it is seen as being highly effective on the students' computational thinking skills self-efficacy, and their computer programming self-efficacy. In addition, high levels of

academic success from mathematics lessons can contribute to students achieving high scores in programming tests. Students and teachers appear to hold similar opinions about the issue of mathematics knowledge contributing to programming knowledge and programming learning.

5.2 Conclusion

This study aimed to investigate seventh-grade students' development of both programming and computational thinking skills through the integration of mathematics into an existing programming course. The focus of the study was to evaluate the effect of teaching programming through an interdisciplinary approach by integrating mathematics with computational thinking skills, mathematic attitude, and programming skills.

First, the study showed that the integration of mathematics as a second discipline in a computer science course played an important role in the improvement of the participants' computational thinking skills. Wing (2008) stated that understanding and finding a creative solution to a problem through the use of computer science techniques was termed as computational thinking. As today's world requires a high degree of ability in utilizing technology, both computer science and computational thinking play an increasingly important role in today's society. Numerous jobs would disappear due to the developments in robotics, and that technology will have a more significant place in modern-day life; opening up more new job areas as a result. Since the importance of computer science and computational thinking is growing alongside and because of these changes to modern industry, developing computational thinking skills in the younger generation has become a matter of some urgency. Therefore, the current study places itself in a significant position because it shows that having computer science education within K-12 curricula develops not only students' programming, problem solving, and algorithmic thinking skills, but also their computational thinking skills as well. Scholars have indicated that the number of opportunities in K-12 education are not sufficiently qualifying or satisfying to

significantly improve students' computational thinking skills (Guzdial, 2015; Kale et al., 2018; Kalelioğlu et al., 2016; Wing, 2008). Since the next generation needs to possess significant computer science knowledge and computational thinking skills, it is of vital importance that K-12 schools, teachers, educators, and administrators should work together to mold curricula in a way that adequately integrates these disciplines. Therefore, the current study has worked to inspire a revised methodology of teaching computer science through its facilitation.

Second, the results of this experimental research with a sample of 188 seventh-grade secondary school students showed that those students who were taught programming through an interdisciplinary approach using mathematics showed a significant increase in their computer programming skills. Researchers agree on the new teaching methodology, which have been studied for almost all disciplines in order to facilitate learning and to make the knowledge gained permanent. This need has increased further in programming learning because of its perceived level of difficulty and complexity, and its inherent requirement for advanced skills such as problem solving, creative thinking, algorithmic thinking, and systematic thinking (Draganoiu et al., 2017). The involvement of a second discipline in K-12 computer science courses can establish a natural link between the real world and the machine world, with learners tasked with thinking not only about the structure and syntaxes involved in programming, but in solving real-world problems as well (Marciuc & Miron, 2017). The relation between computer science and mathematics is undeniable, with both heavily focused on solving problems, following algorithms, and finding solutions for given problems (Burns et al., 2012; Celedón-Pattichis et al., 2013). Therefore, an established need existed to examine the effect of mathematics on programming learning. Hence, the current study aimed to find out whether or not teaching a programming language with the collaboration of mathematics as an interdisciplinary approach affected the participant students' programming skills, and also how it affected their learning retention. The reason for this is suggested that the students receiving the interdisciplinary instructional method had to think more than those receiving the traditional method of instruction, because they also needed to

resolve mathematical problems. Also, the students who were applied the experimental teaching method were challenged with learning not only programming, but also in creating links between mathematics and programming.

Third, the current study showed that the integration of mathematics into programming courses does not positively change students' attitudes towards mathematics as a course. Fábián et al. (2008) stated that teaching mathematics along with the use of technology can have a positive effect on students' mathematics achievement. However, in a study by Hannula (2002), the researcher highlighted that integrating mathematics into a second discipline may negatively affect the students' attitude due to the emotions automatically associated with the concepts of mathematics. The reason why the mathematics attitude of students decreased slightly might be because of their having experienced too many mathematical problems when working on the programming concept. This study recommends that future studies change this in order to modify learners' attitudes to a positive outlook.

Finally, both of the teachers and a selection of the participant students in the current study were interviewed with regards to their opinions about the process. The collected data in the current study showed that teaching programming through the integration of mathematics can have a positive and significant effect on both students' programming and computational thinking skills. This is because both mathematics and programming share the same root aims, which are problem solving, creative thinking, and the following of an algorithm in order to solve a problem.

5.3 Assumptions

When the results of the findings were analyzed, the researcher had an assumption that teachers' attitude effected the study because the MAS results were not expected. Therefore, in order to establish if the teachers' attitudes affected their students' attitudes towards MAS, CTSSES, CPSES and SBPAT, the scores obtained from students were examined according to which of the two teachers had

taught their class. The distribution of the experimental and control groups between the two participating teachers was as follows:

T1: Exp 3, Exp 4, Cont 2, Cont 3

T2: Exp 1, Exp 2, Cont 1, Cont 4

The procedure was same with the quantitative scales' analysis so paired t-test, independent sample t-test and ANCOVA analysis applied to the groups which were separated based on teachers who taught. The results showd that teachers did not have any influence on students' scores. All the results and tables can be found on Appendices F – Teachers' Effect.

5.4 Implications for teachers and instructional designers

The following recommendations are suggested for both teachers and instructional designers in order to create Computer Science lessons that are more appropriate for young learners:

1. A new "Programming" course could be proposed as an independent, supplementary course to the Turkish secondary school curricula;
2. Student motivation could be fostered to reach a higher level through increasing the number of local and national competitions that require the application of programming skills;
3. Problems that are assigned to students should be rewritten rather than being taken direct from course textbooks with the aim to increase students' attitudes towards mathematics.

5.5 Recommendation for Further Research

Based on the results of the current study, the researcher puts forth the following recommendations in terms of future research that could be undertaken in this area of study:

1. Personal variables that may impact on programming education could be studied;
2. The effect of different programming languages and tools on students' computational thinking skills self-efficacy could be studied;
3. Computing and instructional technologies curricula, both in Turkey and internationally could be investigated in order to compare content similarities and differences on the teaching of programming;
4. In addition to the integration of mathematics, courses in both the social sciences and natural sciences could be included in future studies;
5. Plan and execute a long-term project to evaluate the differences between experimental and control groups' programming skills.

5.6 Limitations of the Study

As with all studies, there are certain limitations to the current study. First, the study took longer than was expected due to the nature of working with younger students, and to the events and protocols that schools must adhere to in accordance with the Turkish Ministry of National Education's curricula. Second, it took considerable time to procure all of the relevant ethical permissions from the participants' parents due to the large sample size. Additional time was also required in order to follow the ethical permission as required by the Turkish Ministry of National Education. Third, some of the students were unable to complete the scales applied in the study due to having been absent for an extended period of time due to taking part in certain sporting events. Preparation of the lesson plans took a lot of time because of having

two discipline in one lesson. Last, in working with a private (non-state) school, administering changes to the curriculum was found to be less easy when compared to state schools, which unexpectedly prolonged the duration of the study.

REFERENCES

Aho, A. V. (2012). Computation and Computational Thinking. *The Computer Journal, 55*(7), 832-835. https://doi.org/10.1093/comjnl/bxs074

Akbulut, Y. (2010). *Sosyal bilimlerde SPSS uygulamaları: sık kullanılan istatistiksel analizler ve açıklamalı SPSS çözümleri* [SPSS applications in social sciences: frequently used statistical analysis and annotated SPSS solutions]. İdeal Kültür.

Akçay, T. (2009). *Perceptions of students and teachers about the use of a kid's programming language in computer courses* [Unpublished master's thesis]. Middle East Technical University, Turkey.

Aktümen, M., & Kaçar, A. (2008). Bilgisayar cebiri sistemlerinin matematiğe yönelik tutuma etkisi [Effects of Computer Algebra Systems On Attitudes Towards Mathematics]. *Hacettepe Üniversitesi Eğitim Fakültesi Dergisi, 35*, 13-26.

Alsancak Sirakaya, D. (2019). Programlama Öğretiminin Bilgi İşlemsel Düşünme Becerisine Etkisi [The Effect of Programming Teaching On Computational Thinking]. *Turkish Journal of Social Research/Turkiye Sosyal Arastirmalar Dergisi, 23*(2), 13-26. http://www.efdergi.hacettepe.edu.tr/yonetim/icerik/makaleler/538-published.pdf

Altun, A., & Mazman, S. G. (2012). Programlamaya ilişkin öz yeterlilik algısı ölçeğinin Türkçe formunun güvenirlik ve geçerlik çalışması. *Eğitimde ve Psikolojide Ölçme ve Değerlendirme Dergisi*, *3*(2), 297-308.

Amiri, I. S., & Nikoukar, A. (2011, November 24-25). *Secured Binary Codes Generation for Computer Network Communication* [Conference presentation abstract]. Annual International Conference on Network Technologies & Communication, Singapore.

Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., & Zagami, J. (2016). A K-6 computational thinking curriculum framework: Implications for teacher knowledge. *Journal of Educational Technology & Society*, *19*(3), 47-57. https://www.jstor.org/stable/pdf/jeductechsoci.19.3.47.pdf

Armoni, M. (2011). The nature of CS in K-12 curricula: the roots of confusion. *ACM Inroads, 2*(4), 19-20. https://doi.org/10.1145/2038876.2038883

Atmatzidou, S., & Demetriadis, S. (2016). Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. *Robotics and Autonomous Systems, 75*(B), 661-670. https://doi.org/10.1016/j.robot.2015.10.008

Aydın, G., & Balım, A. G. (2005). Yapılandırmacı yaklaşıma göre modellendirilmiş disiplinler arası uygulama: enerji konularının öğretimi [An Interdisciplinary Application Based on Constructivist Approach: Teaching of Energy Topics]. *Ankara Üniversitesi Eğitim Bilimleri Fakültesi Dergisi, 38*(2), 145-166. https://doi.org/10.1501/Egifak_0000000113

Baldassarri, D., & Abascal, M. (2017). Field experiments across the social sciences. *Annual Review of Sociology, 43*, 41-73. https://doi.org/10.1146/annurev-soc-073014-112445

Barak, M., & Assal, M. (2018). Robotics and STEM learning: Students' achievements in assignments according to the P3 Task Taxonomy—practice, problem-solving, and projects. *International Journal of Technology and Design Education, 28*(1), 121-144. https://doi.org/10.1007/s10798-016-9385-9

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? *Inroads, 2*(1), 48-54. https://doi.org/10.1145/1929887.1929905

Barut, E., Tuğtekin, U., & Kuzu, A. (2016). Programlama eğitiminin bilgi işlemsel düşünme becerileri bağlamında incelenmesi [Investigation of Programming Education in the Context of Computational Thinking Skills]. In *4. Uluslararası Öğretim Teknolojileri ve Öğretmen Eğitimi Sempozyumunda sunulan bildiri* (pp. 210-214). Fırat University, Turkey. http://www.ittes2016.org/belge/oku/37

Battal, A. (2018). *Investigating the use of virtual worlds to teach basics of programming to children: a multiple case study* [Doctoral dissertation, Middle East Technical University]. http://etd.lib.metu.edu.tr/upload/12621793/index.pdf

Battal, A., & Tokel, S. T. (2020). Investigating the Factors Affecting Students' Satisfaction in a Programming Course Designed in 3D Virtual Worlds. *Turkish Online Journal of Qualitative Inquiry, 11*(2), 218-246. https://doi.org/10.17569/tojqi.611707

Baytak, A., & Land, S. M. (2011). An investigation of the artifacts and process of constructing computer games about environmental science in a fifth grade classroom. *Educational Technology Research and Development, 59*, 765-782. http://dx.doi.org/10.1007/s11423-010-9184-z

Becker, H. J. (1993). Teaching with and about computers in secondary schools. *Communications of the ACM, 36*(5), 69-74. https://doi.org/10.1145/155049.155066

Beecher, K. (2017). *Computational Thinking*. BCS.

Bektik, D. (2017). *Learning Analytics for Academic Writing through Automatic Identification of Meta-discourse* [Doctoral dissertation, The Open University]. https://doi.org/10.21954/ou.ro.0000c1f9

Belanger, C., Christenson, H., & Lopac, K. (2018). *Confidence and common challenges: the effects of teaching computational thinking to students ages 10-16.* St. Catherine University, MN. https://sophia.stkate.edu/maed/267/

Benitti, F. B. V., & Spolaôr, N. (2017). How have robots supported stem teaching? In M. S. Khine (Ed.), *Robotics in STEM Education* (pp. 103-129). Springer.

Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. *ACM SIGCSE Bulletin, 39*(2), 32-36. https://doi.org/10.1145/1272848.1272879

Benton, L., Hoyles, C., Kalas, I., & Noss, R. (2017). Bridging primary programming and mathematics: Some findings of design research in England. *Digital Experiences in Mathematics Education, 3*(2), 115-138. https://doi.org/10.1007/s40751-017-0028-x

Berland, M., & Lee, V. R. (2011). Collaborative strategic board games as a site for distributed computational thinking. *International Journal of Game-Based Learning (IJGBL), 1*(2), 65-81. https://doi.org/10.4018/ijgbl.2011040105

Berry, M. (2015). *QuickStart: Computing Primary Handbook.* BCS.

Bhatt, A. (2010). Evolution of clinical research: a history before and beyond James Lind. *Perspectives in Clinical Research, 1*(1), 6-10. https://www.picronline.org/text.asp?2010/1/1/6/71839

Bilge Kunduz. (2018). *2018 Yılı Uygulama Sonuçları* [The results of 2018 Application]. http://www.bilgekunduz.org/wp-content/uploads/2019/04/bilgekunduz_2018_v4.pdf

Birt, L., Scott, S., Cavers, D., Campbell, C., & Walter, F. (2016). Member checking: a tool to enhance trustworthiness or merely a nod to validation? *Qualitative health research*, *26*(13), 1802-1811. https://doi.org/10.1177%2F1049732316654870

Bishop-Clark, C., Courte, J., & Howard, E. (2007). A quantitative and qualitative investigation of using ALICE programming to improve confidence, enjoyment and achievement among non-majors. *Journal of Educational Computing Research, 37*(2), 193-207. https://doi.org/10.2190%2FJ8W3-74U6-Q064-12J5

Bornat, R., & Dehnadi, S. (2008). Mental models, consistency and programming aptitude. In S. Hamilton & M. Hamilton (Eds.), *Proceedings of the Tenth Conference on Australasian Computing Education* (Vol. 78, pp. 53-61). Australian Computer Society. https://dl.acm.org/doi/epdf/10.5555/1379249.1379253

Borrego, M., & Newswander, L. K. (2010). Definitions of interdisciplinary research: Toward graduate-level interdisciplinary learning outcomes. *The Review of Higher Education, 34*(1), 61-84. https://doi.org/10.1353/rhe.2010.0006

Brennan, K. (2017). Computer Programming. In K. Peppler (Ed.), *SAGE Encyclopedia of Out-of-School Learning* (Vol. 1, pp. 123-126). Sage. https://doi.org/10.4135/9781483385198.n55

Brennan, K., & Resnick, M. (2012, April 13-17). *Using artifact-based interviews to study the development of computational thinking in interactive media design* [Paper presentation]. American Educational Research Association Meeting, Vancouver, BC, Canada. https://web.media.mit.edu/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf

Bruce, K. B., Drysdale, R. L. S., Kelemen, C., & Tucker, A. (2003). Why math? *Communications of the ACM, 46*(9), 40-44. https://doi.org/10.1145/903893.903918

Burke, Q., & Kafai, Y. B. (2010). Programming & storytelling: opportunities for learning about coding & composition. In *Proceedings of the 9th International Conference on Interaction Design and Children* (pp. 348-351). https://doi.org/10.1145/1810543.1810611

Burns, R., Pollock, L., & Harvey, T. (2012). Integrating hard and soft skills: software engineers serving middle school teachers. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (pp. 209-214). ACM.

Büyüköztürk, Ş., Çakmak, E. K., Akgün, Ö. E., Karadeniz, Ş., & Demirel, F. (2010). *Bilimsel araştırma yöntemleri* [Scientific Research Methods] (5th ed.). Pegem Akademi.

Cai, W., & Sankaran, G. (2015). Promoting critical thinking through an interdisciplinary study abroad program. *Journal of International Students, 5*(1), 38-49. https://doi.org/10.32674/jis.v5i1

Calao, L. A., Moreno-León, J., Correa, H. E., & Robles, G. (2015). Developing mathematical thinking with scratch. In G. Conole, T. Klobučar, C. Rensing, J. Konert, & E. Lavoué (Eds.), *Design for teaching and learning in a networked world* (pp. 17-27). Springer. https://doi.org/10.1007/978-3-319-24258-3_2

Calder, N. (2010). Using Scratch: An integrated problem-solving approach to mathematical thinking. *Australian Primary Mathematics Classroom, 15*(4), 9-14.

Cansu, S. K., & Cansu, F. K. (2019). An Overview of Computational Thinking. *International Journal of Computer Science Education in Schools, 3*(1). https://doi.org/10.21585/ijcses.v3i1.53

Caspersen, M. E., & Kolling, M. (2009). STREAM: A first programming process. *ACM Transactions on Computing Education, 9*(1), Article 4. https://doi.org/10.1145/1513593.1513597

Cassel, L. N. (2011). Interdisciplinary computing is the answer: now, what was the question? *ACM Inroads, 2*(1), 4-6. https://doi.org/10.1145/1929887.1929888

Castro, F. G., Kellison, J. G., Boyd, S. J., & Kopak, A. (2010). A methodology for conducting integrative mixed methods research and data analyses. *Journal of Mixed Methods Research, 4*(4), 342-360. https://doi.org/10.1177%2F1558689810382916

Celedón-Pattichis, S., LópezLeiva, C. A., Pattichis, M. S., & Llamocca, D. (2013). An interdisciplinary collaboration between computer engineering and mathematics/bilingual education to develop a curriculum for underrepresented middle school students. *Cultural Studies of Science Education, 8*(4), 873-887. https://doi.org/10.1007/s11422-013-9516-5

Çelik, A., & Özdener, N. (2019). Bilgisayarlı Ve Bilgisayarsız Programlama Etkinliklerinin Güdülenme Üzerindeki Etkisi [The Effect of Plugged and Unplugged Programming Activities on Motivation]. *Akademik Sosyal Araştırmalar Dergisi, 7*(88), 651-669. http://dx.doi.org/10.16992/ASOS.14692

Çetin, O., & Günay, Y. (2011). Fen eğitimine yönelik örnek bir web tabanlı öğretim materyalinin hazırlanması ve bu materyalin öğretmen öğrenci görüşleri doğrultusunda değerlendirilmesi [Preparation of a Web-Based Teaching Material Designed for Science Education and Evaluation of This Material According to Teachers' and Students' Views]. *Ahi Evran Üniversitesi Eğitim Fakültesi Dergisi, 12*(2), 175-202.

Çetinkaya, H. N. (2019). *Bilişim teknolojileri ve yazılım dersindeki etkinliklerin bilgi işlemsel düşünme ve bazı değişkenler açısından incelenmesi* [An investigation of activities in information technology and software course in terms of computational thinking and some variables]. İnönü University, Turkey.

Cheng, H. (1972). Constructive mathematics and computer science. In *Proceedings of the ACM Annual Conference* (Vol. 2, pp. 986-990). ACM.

Chilcott, L. (Director) (2013). *CodeStars* [short film]. Grommit.

Çınar, M., & Tüzün, H. (2017, May 26). *Eğitimde bilgisayımsal düşünme uygulamalarına ilişkin bir alanyazın incelemesi* [Paper presentation]. 11th Uluslararası Bilgisayar ve Öğretim Teknolojileri Eğitimi Sempozyumunda sunulan bildiri. İnöünü University, Malatya, Turkey.

Clements, D. H., & Gullo, D. F. (1984). Effects of computer programming on young children's cognition. *Journal of Educational Psychology, 76*(6), 1051-1058. https://psycnet.apa.org/doi/10.1037/0022-0663.76.6.1051

Cohen, B., Smith, B., & Mitchell, R. (2008). Toward a sustainable conceptualization of dependent variables in entrepreneurship research. *Business Strategy and the Environment*, *17*(2), 107-119. https://doi.org/10.1002/bse.505

Conrod, P., & Tylee, L. (2013). *Computer Bible Games for Microsoft Small Basic*. BibleByte.

Cooper, S., Dann, W., & Pausch, R. (2003). Teaching object-first in introductory computer science. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education* (pp. 191-195). ACM.

Corradini, I., Lodi, M., & Nardelli, E. (2017). Computational thinking in Italian schools: quantitative data and teachers' sentiment analysis after two years of "Programma il Futuro" project. In R Davoli, (Ed.), *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 224-229). ACM.

Creswell, J. W. (1994). *Research design: Qualitative and quantitative approaches*. Sage.

Creswell, J. W. (2012). *Educational research: Planning, conducting, and evaluating quantitative and qualitative research* (4th ed.). Pearson.

Creswell, J. W. (2013). *Research design: Qualitative, quantitative, and mixed methods approaches* (2nd ed.). Sage.

Creswell, J. W., & Clark, V. L. P. (2007). *Designing and Conducting Mixed Methods Research*. Sage.

Creswell, J. W., & Miller, D. L. (2000). Determining Validity in Qualitative Inquiry. *Theory Into Practice*, *25*(3), 124-130. https://doi.org/10.1207/s15430421tip3903_2

Curry, L. A., Nembhard, I. M., & Bradley, E. H. (2009). Qualitative and mixed methods provide unique contributions to outcomes research. *Circulation*, *119*(10), 1442-1452. https://doi.org/10.1161/CIRCULATIONAHA.107.742775

Czerkawski, B. C., & Lyman, E. W., III. (2015). Exploring issues about computational thinking in higher education. *TechTrends*, *59*(2), 57-65. http://dx.doi.org/10.1007/s11528-015-0840-3.

Dagdilelis, V., Satratzemi, M., & Evangelidis, G. (2004). Introducing secondary education students to algorithms and programming. *Education and Information Technologies*, *9*(2), 159-173. https://doi.org/10.1023/B:EAIT.0000027928.94039.7b

Daily, S. B., Leonard, A. E., Jörg, S., Babu, S., Gundersen, K., & Parmar, D. (2015). Embodying computational thinking: initial design of an emerging technological learning tool. *Technology, Knowledge and Learning*, *20*(1), 79-84. https://doi.org/10.1007/s10758-014-9237-1

Davenport, J. H., Wilson, D., Graham, I., Sankaran, G., Spence, A., Blake, J., & Kynaston, S. (2014). Interdisciplinary teaching of computing to mathematics students: Programming and discrete mathematics. *MSOR Connections, 14*(1), 1-8. https://doi.org/10.11120/msor.2014.00021

De Jesus Gomes, A., Mendes, A. J., & Marcelino, M. J. (2015). Computer Science Education Research: An Overview and Some Proposals. In R. Queirós (Ed.), *Innovative Teaching Strategies and New Learning Paradigms in Computer Programming* (pp. 1-29). IGI Global. https://doi.org/10.4018/978-1-4666-7304-5.ch001

Denning, P. J. (2009). The profession of IT Beyond computational thinking. *Communications of the ACM, 52*(6), 28-30. https://doi.org/10.1145/1516046.1516054

Denning, P. J., Tedre, M., & Yongpradit, P. (2017). Misconceptions about Computer Science. *Communications of the ACM, 60*(3), 31-33. https://doi.org/10.1145/3041047

Denzin, N. K., & Lincoln, Y. S. (Eds.). (2011). *The Sage handbook of qualitative research*. Sage.

De Raadt, M., Watson, R., & Toleman, M. (2002). Language trends in introductory programming courses. In *Proceedings of the 2002 Informing Science+ Information Technology Education Joint Conference (InSITE 2002)* (pp. 229-337). Informing Science Institute.

Djambong, T., & Freiman, V. (2016). Task-Based Assessment of Students' Computational Thinking Skills Developed through Visual Programming or Tangible Coding Environments. In D. G. Sampson, J. M. Spector, D. Ifenthaler, & P. Isaias (Eds.), *Proceedings of the International Association for Development of the Information Society (IADIS) International Conference on Cognition and Exploratory Learning in the Digital Age (CELDA)* (pp. 41-51). IADIS.

Draganoiu, R., Moldoveanu, A., & Braescu, A. (2017). The Math of Programming-Interdisciplinary Approach. In I. Roceanu, J. Rehrl, L. Ciolan, I. Stefan, C. Radu, & B. Logofatu (Eds.), *The International Scientific Conference eLearning and Software for Education* (Vol. 2, pp. 473-481). Carol I National Defense University. https://doi.org/10.12753/2066-026X-17-152

Duatepe, A., & Çilesiz, Ş. (1999). Matematik tutum ölçeği geliştirilmesi [Developing Mathematics Attittue Scale]. *Hacettepe Üniversitesi Eğitim Fakültesi Dergisi, 16*(16). http://www.efdergi.hacettepe.edu.tr/yonetim/icerik/makaleler/1202-published.pdf

Duncan, C., Bell, T., & Tanimoto, S. (2014). Should your 8-year-old learn coding? In C. Schulte (Ed.), *Proceedings of the 9th Workshop in Primary and Secondary Computing Education* (pp. 60-69). ACM.

Esteves, M., Fonseca, B., Morgado, L., & Martins, P. (2008). Contextualization of programming learning: A virtual environment study. In *2008 38th Annual Frontiers in Education Conference* (pp. F2A-17). IEEE.

Fábián, Á., Győrbíró, N., & Hományi, G. (2008). Activity recognition system for mobile phones using the Motion Band device. In *Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications* (Article 41). ICST.

Feaster, Y., Segars, L., Wahba, S. K., & Hallstrom, J. O. (2011). Teaching CS unplugged in the high school (with limited success). In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education* (pp. 248-252). ACM.

Fesakis, G., & Serafeim, K. (2009). Influence of the familiarization with Scratch on future teachers' opinions and attitudes about programming and ICT in education. *ACM SIGCSE, 41*(3), 258-262. https://doi.org/10.1145/1595496.1562957

Fields, D. A., Lui, D., & Kafai, Y. B. (2019). Teaching computational thinking with electronic textiles: Modeling iterative practices and supporting personal projects in exploring computer science. In S.-C. Kong & H. Abelson (Eds.), *Computational Thinking Education* (pp. 279-294). Springer. https://doi.org/10.1007/978-981-13-6528-7_16

Fincher, S., Cooper, S., Kölling, M., & Maloney, J. (2010). Comparing alice, greenfoot & scratch. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (pp. 192-193). ACM.

Fırat, S. (2011). *Bilgisayar destekli eğitsel oyunlarla gerçekleştirilen matematik öğretiminin kavramsal öğrenmeye etkisi* [The effect of mathematics teaching performed through educational computer games on conceptual learning] [Master's thesis, Adıyaman University, Turkey]. http://dspace.adiyaman.edu.tr:8080/xmlui/handle/20.500.12414/92

Fisler, K., Schanzer, E., Weimar, S., Fetter, A., Renninger, K. A., Krishnamurthi, S., Politz, J. G., Lerner, B., Poole, J., & Koerner, C. (2020). Evolving a K-12 Curriculum for Integrating Computer Science into Mathematics. *Circles, 8*(4), Article 2.

Fojtik, R. (2014). Design patterns in the teaching of programming. *Procedia - Social and Behavioral Sciences, 143*, 352-357. https://doi.org/10.1016/j.sbspro.2014.07.493

Francis, K., & Davis, B. (2018). Coding Robots as a Source of Instantiations for Arithmetic. *Digital Experiences in Mathematics Education*, *4*(2-3), 71-86. https://doi.org/10.1007/s40751-018-0042-7

Gadanidis, G. (2017). Artificial intelligence, computational thinking, and mathematics education. The International Journal of Information and Learning Technology, 34(2), 133–139. doi:10.1108/ijilt-09-2016-0048

Gadanidis, G., Cendros, R., Floyd, L., & Namukasa, I. (2017). Computational thinking in mathematics teacher education. Contemporary Issues in Technology and Teacher Education, 17(4), 458-477.

Gadanidis, G. (2017). Five affordances of computational thinking to support elementary mathematics education. Journal of Computers in Mathematics and Science Teaching 36(2), 143-151.

Gal-Ezer, J., & Stephenson, C. (2014). A tale of two countries: Successes and challenges in K-12 computer science education in Israel and the United States. *ACM Transactions on Computing Education (TOCE)*, *14*(2), Article 8. https://doi.org/10.1145/2602483

Gerecke, U., & Wagner, B. (2007). The challenges and benefits of using robots in higher education. *Intelligent Automation and Soft Computing*, *13*(1), 29-43. https://doi.org/10.1080/10798587.2007.10642948

Göksoy, S., & Yılmaz, İ. (2018). Bilişim teknolojileri öğretmenleri ve öğrencilerinin robotik ve kodlama dersine ilişkin görüşleri [The Opinions of Information Relations Teacher and their Students with Regard to Lessons of Robots and Decoding]. *Düzce Üniversitesi Sosyal Bilimler Enstitüsü Dergisi*, *8*(1), 178-196. https://dergipark.org.tr/en/download/article-file/517224

Goldsmith, A. H., Hamilton, D., Hornsby, K., & Wells, D. (2018, May 7). *Interdisciplinary Approaches to Teaching*. https://serc.carleton.edu/sp/library/interdisciplinary/index.html .

Good, J. (2011). Learners at the wheel: Novice programming environments come of age. *International Journal of People-Oriented Programming (IJPOP)*, *1*(1), 1-24. https://doi.org/10.4018/ijpop.2011010101

Gorman, H., & Bourne, L. E. (1983). Learning to think by learning LOGO: Rule learning in third-grade computer programmers. *Bulletin of the Psychonomic Society*, *21*(3), 165-167. https://doi.org/10.3758/BF03334676

Graham, K. J., & Fennell, F. (2001). Principles and standards for school mathematics and teacher education: Preparing and empowering teachers. *School Science and Mathematics*, *101*(6), 319-327. https://doi.org/10.1111/j.1949-8594.2001.tb17963.x

Gribbons, B., & Herman, J. (1996). True and quasi-experimental designs. *Practical Assessment, Research, and Evaluation, 5*(1), Article 14. https://doi.org/10.7275/fs4z-nb61

Grover, S., Cooper, S., & Pea, R. (2014). Assessing computational learning in K-12. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education* (pp. 57-62). ACM.

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher, 42*(1), 38-43. https://doi.org/10.3102%2F0013189X12463051

Gülbahar, Y., Kert, S. B., & Kalelioğlu, F. (2019). Bilgi işlemsel düşünme becerisine yönelik öz yeterlik algısı ölçeği: geçerlik ve güvenirlik çalışması [The Self-Efficacy Perception Scale for Computational Thinking Skill: Validity and Reliability Study]. *Türk Bilgisayar ve Matematik Eğitimi Dergisi*, *10*(1), 1-29. https://doi.org/10.16949/turkbilmat.385097

Güleryüz, B. G. (2019). *Ortaokul öğrencilerinin ders içi robotik kodlama etkinliklerinin blok tabanlı programlamaya ilişkin öz yeterlilik algısına etkisi ve robotik kodlama hakkındaki görüşleri* [Effect of secondary school students' the self-efficacy on incourse robotics coding activities of block based programming and their opinions on robotics coding] [Master's thesis]. Bursa Uludağ University, Turkey.

Gürbüz, R., & Birgin, O. (2012). The effect of computer-assisted teaching on remedying misconceptions: The case of the subject "probability". *Computers & Education, 58*(3), 931-941. https://doi.org/10.1016/j.compedu.2011.11.005

Guzdial, M. (2004). Programming environments for novices. In S. Fincher & M. Petre (Eds.), *Computer Science Education Research* (pp. 127-154). Routledge Falmer.

Guzdial, M. (2015). Learner-centered design of computing education: Research on computing for everyone. In *Synthesis Lectures on Human-Centered Informatics*, *8*(6), 1-165. https://doi.org/10.2200/S00684ED1V01Y201511HCI033

Guzdial, M., & DiSalvo, B. (2013). Computing education: Beyond the classroom. *Computer, 46*(9), 30-31. https://doi.org/10.1109/MC.2013.306

Guzdial, M., & Forte, A. (2005). Design process for a non-majors computing course. *ACM SIGCSE Bulletin, 37*(1), 361-365. https://doi.org/10.1145/1047124.1047468

Guzdial, M., & Morrison, B. (2016). Growing computer science education into a STEM education discipline. *Communications of the ACM, 59*(11), 31-33. http://dx.doi.org/10.1145/3000612

Hancock, R. G. (2004). Experimental, quasi-experimental and nonexperimental design and analysis with latent variables. In D. Kaplan (Ed.), *The Sage Handbook of Quantitative Methodology for the Social Sciences* (pp. 91-105). Sage.

Hannula, M. S. (2002). Attitude towards mathematics: Emotions, expectations and values. *Educational Studies in Mathematics, 49*(1), 25-46. https://doi.org/10.1023/A:1016048823497

Harrell, M. C., & Bradley, M. A. (2009). *Data collection methods. Semi-structured interviews and focus groups.* Rand National Defense Research Institute. https://www.rand.org/pubs/technical_reports/TR718.html

Harvie, D. P., Estes, T., & Major, K. (2018). Use of Commercial Online Training to Augment Programming Language Education. In *Proceedings of the 19th Annual SIG Conference on Information Technology Education (SIGITE '18)* (p. 89). ACM. https://doi.org/10.1145/3241815.3241832

Haseski, H. I., Ilic, U., & Tugtekin, U. (2018). Defining a New 21st Century Skill-Computational Thinking: Concepts and Trends. *International Education Studies, 11*(4), 29-42. https://doi.org/10.5539/ies.v11n4p29

Hassan, Z. A., Schattner, P., & Mazza, D. (2006). Doing a pilot study: why is it essential? *Malaysian Family Physician, 1*(2-3), 70-73. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4453116/

Havill, J. T., & Ludwig, L. D. (2007). Technically speaking: fostering the communication skills of computer science and mathematics students. *ACM SIGCSE Bulletin, 39*(1), 185-189. https://doi.org/10.1145/1227504.1227375

Hemmendinger, D. (2010). A plea for modesty. *ACM Inroads, 1*(2), 4-7. https://doi.org/10.1145/1805724.1805725

Herbsleb, J. D. (2005). Beyond computer science. In *Proceedings of the 27th International Conference on Software Engineering* (pp. 23-27). https://doi.org/10.1145/1062455.1062466

Hickmott, D., Prieto-Rodriguez, E., & Holmes, K. (2018). A scoping review of studies on computational thinking in K–12 mathematics classrooms. *Digital Experiences in Mathematics Education, 4*(1), 48-69. https://doi.org/10.1007/s40751-017-0038-8

Holmboe, C., McIver, L., & George, C. E. (2001). Research Agenda for Computer Science Education. In *Annual Workshop of the Psychology of Programming Interest Group (PPIG)* (Vol. 13, pp. 207-223). Sheffield Hallam University.

Howland, K., Good, J., & Nicholson, K. (2009). Language-based support for computational thinking. In R. DeLine, M. Minas, & M. Erwig (Eds.), *Proceedings: 2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 147-150). IEEE.

Huck, S. W. (2008). *Reading statistics and research*. Allyn & Bacon.

International Society for Technology in Education & Computer Science Teachers Association. (2011). *Operational definition of computational thinking for K-12 education*. https://csta.acm.org/Curriculum/sub/CurrFiles/CompThinkingFlyer.pdf

Ioannidou, A., Bennett, V., Repenning, A., Koh, K. H., & Basawapatna, A. (2011, April 8-12). *Computational Thinking Patterns* [Paper presentation]. Annual Meeting of the American Educational Research Association, New Orleans, LA.

Israel, M., Pearson, J. N., Tapia, T., Wherfel, Q. M., & Reese, G. (2015). Supporting all learners in school-wide computational thinking: A cross-case qualitative analysis. *Computers & Education*, *82*, 263-279. https://doi.org/10.1016/j.compedu.2014.11.022

Iyer, S. (2019). Teaching-Learning of Computational Thinking in K-12 Schools in India. In S.-C. Kong & H. Abelson (Eds.), *Computational Thinking Education* (pp. 363-382). Springer.

Jaccheri, L., & Sindre, G. (2007). Software engineering students meet interdisciplinary project work and art. In E. Banissi, R. Aslak Burkhard, G. Grinstein, L. Stuart, T. G. Wyeld, G. Andrienko, J. Dykes, M. Jern, A. Faiola, D. Groth, A. Ursyn, A. J. Cowell, & M. Hou (Eds.), *11th International Conference Information Visualization (IV'07)* (pp. 925-934). IEEE. https://doi.org/10.1109/IV.2007.102

Jackson, M., & Cox, D. R. (2013). The principles of experimental design and their application in sociology. *Annual Review of Sociology*, *39*, 27-49. https://doi.org/10.1146/annurev-soc-071811-145443

Jehlička, V. (2010). Interdisciplinary relations in teaching of programming. In *Proceedings of the 2010 International Conference on Applied Computing* (pp. 33-38). World Scientific and Engineering Academy and Society (WSEAS).

Jenkins, T. (2002). On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences* (Vol. 4, pp. 53-58).

Jones, C. (2010). Interdisciplinary approach-advantages, disadvantages, and the future benefits of interdisciplinary studies. *Essai, 7*(1), Article 26. http://dc.cod.edu/essai/vol7/iss1/26

Jun, S.-J., Han, S.-K., & Kim, S.-H. (2017). Effect of design-based learning on improving computational thinking. *Behaviour and Information Technology, 36*(1), 43-53. https://doi.org/10.1080/0144929X.2016.1188415

Kafai, Y., & Burke, Q. (2014). *Connected code : why children need to learn programming*. MIT.

Kale, U., Akcaoglu, M., Cullen, T., Goh, D., Devine, L., Calvert, N., & Grise, K. (2018). Computational what? Relating computational thinking to teaching. *TechTrends, 62*(6), 574-584. https://doi.org/10.1007/s11528-018-0290-9

Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: Code. org. *Computers in Human Behavior, 52*, 200-210. https://doi.org/10.1016/j.chb.2015.05.047

Kalelioğlu, F., & Gülbahar, Y. (2014). The Effects of Teaching Programming via Scratch on Problem Solving Skills. *Informatics in Education, 13*(1), 33-50. https://www.ceeol.com/search/article-detail?id=123695

Kalelioğlu, F., Gülbahar, Y., & Kukul, V. (2016). A framework for computational thinking based on a systematic research review. *Baltic Journal of Modern Computing, 4*(3), 583-596. http://hdl.handle.net/11727/3831

Karabak, D., & Güneş, A. (2013). Ortaokul birinci sınıf öğrencileri için yazılım geliştirme alanında müfredat önerisi [Curriculum Proposal for First Class Secondary School Students in the Field of Software Development]. *Eğitim ve Öğretim Araştırmaları Dergisi, 2*(3), 163-169. http://jret.org/FileUpload/ks281142/File/21.karabak.pdf

Kasalak, İ. (2017). *Robotik Kodlama Etkinliklerinin Ortaokul Öğrencilerinin Kodlamaya İlişkin Özyeterlik Algılarına Etkisi Ve Etkinliklere İlişkin Öğrenci Yaşantıları* [Effects of Robotic Coding Activities on The Effectiveness Of Secondary School Students' Self-Efficacy and Student Experience About Activities] [Master's thesis]. Hacettepe University, Turkey.

Kaucic, B., & Asic, T. (2011). Improving introductory programming with Scratch. In P. Biljanovic, K. Skala, S. Golubić, N. Bogunović, S. Ribarić, M. Čičin-Šain, D. Čišić, Ž. Hutinski, M. Baranović, M. Mauher, & L. Ordanić (Eds.), *Proceedings of the 34th International Convention* (pp. 1095-1100). IEEE.

Kazez, H., & Genç, Z. (2016a). İlkokul Matematik Öğretiminde Yeni Bir Yaklaşım: Lego Moretomath [A New Approach to Elementary Mathematics Teaching: Lego Moretomath]. *Journal of Instructional Technologies & Teacher Education, 5*(2), 59-71. https://dergipark.org.tr/en/pub/jitte/issue/25090/264800

Kazez, H., & Genç, Z. (2016b). Research Trends in Lego and Robotic Usage in Education: A Document Analysis. *Journal of Instructional Technologies & Teacher Education, 5*(1), 19-31. https://dergipark.org.tr/en/pub/jitte/issue/25089/264784

Kazimoglu, C., Kiernan, M., Bacon, L., & MacKinnon, L. (2012). Learning Programming at the Computational Thinking Level via Digital Game-Play.

Procedia Computer Science, 9, 522-531. https://doi.org/10.1016/j.procs.2012.04.056

Kebritchi, M., Hirumi, A., & Bai, H. (2010). The effects of modern mathematics computer games on mathematics achievement and class motivation. *Computers & Education, 55*(2), 427-443. https://doi.org/10.1016/j.compedu.2010.02.007

Kelleher, C., & Pausch, R. (2007). Using storytelling to motivate programming. *Communications of the ACM, 50*(7), 58-64. https://doi.org/10.1145/1272516.1272540

Kılıç, S., Gökoğlu, S., & Öztürk, M. (2020). A Valid and Reliable Scale for Developing Programming-Oriented Computational Thinking. *Journal of Educational Computing Research* (Advanced online publication). https://doi.org/10.1177%2F0735633120964402

Kivits, J. (2005). Online interviewing and the research relationship. In C. Hine (Ed.), *Virtual methods; Issues in social research on the Internet* (pp. 35-50). Berg.

Klassen, M. (2006). Visual approach for teaching programming concepts. In *Proceedings of the 9th International Conference on Engineering Education* (pp. 141-145). ACM.

Knuth, D. E. (1974). Computer science and its relation to mathematics. *The American Mathematical Monthly, 81*(4), 323-343. https://doi.org/10.1080/00029890.1974.11993556

Köksal, M. S. (2013). Fen Eğitiminde Yapılan Deneysel Çalışmalar için Geniş Kapsamlı Bir Araştırma Deseni [A comprehensive research design for experimental studies in science education]. *Elementary Education Online, 12*(3), 628-634. http://ilkogretim-online.org.tr/index.php/io/article/view/1380

Kraemer, H. C., Mintz, J., Noda, A., Tinklenberg, J., & Yesavage, J. A. (2006). Caution regarding the use of pilot studies to guide power calculations for study proposals. *Archives of General Psychiatry, 63*(5), 484-489. https://doi.org/10.1001/archpsyc.63.5.484

Kramarski, B., & Zeichner, O. (2001). Using technology to enhance mathematical reasoning: Effects of feedback and self-regulation learning. *Educational Media International, 38*(2-3), 77-82. https://doi.org/10.1080/09523980110041458

Kurilovas, E., & Dagiene, V. (2016). Computational thinking skills and adaptation quality of virtual learning environments for learning informatics. *International Journal of Engineering Education, 32*(4), 1596-1603. https://www.ijee.ie/contents/c320416.html

Kvale, S. (1983). The qualitative research interview: A phenomenological and a hermeneutical mode of understanding. *Journal of Phenomenological Psychology, 14*(1-2), 171-196. https://doi.org/10.1163/156916283X00090

Kwon, D.-Y., Kim, H.-S., Shim, J.-K., & Lee, W.-G. (2012). Algorithmic bricks: a tangible robot programming tool for elementary school students. *Education IEEE Transactions, 55*(4), 474-479. https://doi.org/10.1109/TE.2012.2190071

Lancaster, G. A., Dodd, S., & Williamson, P. R. (2004). Design and analysis of pilot studies: recommendations for good practice. *Journal of Evaluation in Clinical Practice, 10*(2), 307-312. https://doi.org/10.1111/j..2002.384.doc.x

LeBlanc, M. D., & Leibowitz, R. (2006). Discrete partnership: a case for a full year of discrete math. *ACM SIGCSE Bulletin, 38*(1), 313-317. https://doi.org/10.1145/1121341.1121438

LeCompte, M. D., & Goetz, J. P. (1982). Problems of reliability and validity in ethnographic research. *Review of Educational Research, 52*(1), 31-60. https://doi.org/10.3102%2F00346543052001031

Lee, I., Grover, S., Martin, F., Pillai, S., & Malyn-Smith, J. (2019). Computational Thinking from a Disciplinary Perspective: Integrating Computational Thinking in K-12 Science, Technology, Engineering, and Mathematics Education. Journal of Science Education and Technology, 29(1), 1–8. doi:10.1007/s10956-019-09803-w

Leech, N. L., & Onwuegbuzie, A. J. (2009). A typology of mixed methods research designs. *Quality & Quantity, 43*(2), 265-275. https://doi.org/10.1007/s11135-007-9105-3

Lin, C.-H., Liu, E. Z.-F, Kou, C.-H, Virnes, M., Sutinen, E., & Cheng, S.-S. (2009). A case analysis of creative spiral instruction model and students' creative problem-solving performance in a Lego robotics course. In M. Chang, R. Kuo, Kinshuk, G.-D. Chen, & M. Hirose (Eds.), *Edutainment '09: Proceedings of the 4th International Conference on E-Learning and Games: Learning by Playing* (pp. 501-505). Springer. https://doi.org/10.1007/978-3-642-03364-3_61

Lin, Y.-T., Wang, M.-T., & Wu, C.-C. (2019). Design and Implementation of Interdisciplinary STEM Instruction: Teaching Programming by Computational Physics. *Asia-Pacific Education Researcher, 28*(1), 77-91. https://doi.org/10.1007/s40299-018-0415-0

Lincoln, Y. S., & Guba, E. G. (1985). *Naturalistic inquiry*. Sage.

Liu, E. Z.-F, Lin, C.-H., Liou, P.-Y., Feng, H.-C., & Hou, H.-T. (2013). An analysis of teacher-student interaction patterns in a robotics course for kindergarten children: A pilot study. *Turkish Online Journal of Educational Technology, 12*(1), 9-18. http://www.tojet.net/articles/v12i1/1212.pdf

Loewen, S., & Plonsky, L. (2015). *An A–Z of applied linguistics research methods*. Macmillan International Higher Education.

Lou, S.-J., Shih, R.-C., Diez, C. R., & Tseng, K.-H. (2011). The impact of problem-based learning strategies on STEM knowledge integration and attitudes: an exploratory study among female Taiwanese senior high school students. *International Journal of Technology and Design Education, 21*(2), 195-215. https://doi.org/10.1007/s10798-010-9114-8

Lu, J. J., & Fletcher, G. H. (2009). Thinking about computational thinking. *ACM SIGCSE Bulletin, 41*(1), 260-264. https://doi.org/10.1145/1508865.1508959

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. https://doi.org/10.1016/j.chb.2014.09.012

Mahadeo, J., Hazari, Z., & Potvin, G. (2020). Developing a Computing Identity Framework: Understanding Computer Science and Information Technology Career Choice. *ACM Transactions on Computing Education (TOCE), 20*(1), Article 7. https://doi.org/10.1145/3365571

Mahadev, A., & Conner, S. (2014). An interdisciplinary approach to teaching a computer ethics course. *Journal of Computing Sciences in Colleges, 29*(5), 202-207. https://dl.acm.org/doi/10.5555/2600623.2600663

Malan, D. J., & Leitner, H. H. (2007). Scratch for Budding Computer Scientists. In *SIGCSE '07: Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education* (pp. 223-227). ACM. https://doi.org/10.1145/1227504.1227388

Maloney, J., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by choice: urban youth learning programming with scratch. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education (SIGCSE '08)* (pp. 367-371). ACM. https://doi.org/10.1145/1352135.1352260

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE), 10*(4), Article 16. https://doi.org/10.1145/1868358.1868363

Mann, C., & Stewart, F. (2000). *Internet communication and qualitative research: A handbook for researching online*. Sage.

Marciuc, D., & Miron, C. (2017). Using Vpython Programming for Interdisciplinary Learning and Problem Solving. *eLearning & Software for Education, 2*, 106-113. https://www.ceeol.com/search/article-detail?id=538758

Mata, M. D. L., Monteiro, V., & Peixoto, F. (2012). Attitudes towards mathematics: Effects of individual, motivational, and social support factors. *Child*

Development Research, 2012, Article 876028. http://dx.doi.org/10.1155/2012/876028

Mazman, S. G., & Altun, A. (2013). Programlama – I Dersinin BÖTE Bölümü Öğrencilerinin Programlamaya İlişkin Öz Yeterlilik Algıları Üzerine Etkisi [The effect of introductory to programming course on programming self-efficacy of CEIT students]. *Journal of Instructional Technologies & Teacher Education, 2*(3), 24-29. https://dergipark.org.tr/en/pub/jitte/issue/25082/264710

Mehra, B. (2002). Bias in qualitative research: Voices from an online classroom. *The Qualitative Report, 7*(1), 1-19. https://nsuworks.nova.edu/tqr/vol7/iss1/2

Merriam, S. B., & Tisdell, E. J. (2015). *Qualitative research: A guide to design and implementation*. John Wiley & Sons.

Mertler, C. A. (2019). Quantitative Methodology in Adolescent Research. *The Encyclopedia of Child and Adolescent Development*, 1-14.

Mészárosová, E. (2015). Is Python an Appropriate Programming Language for Teaching Programming in Secondary Schools? *International Journal of Information and Communication Technologies in Education, 4*(2), 5-14. https://doi.org/10.1515/ijicte-2015-0005

Miceli, R., Civario, G., Sikora, A., César, E., Gerndt, M., Haitof, H., Navarrete, C., Benkner, S., Sandrieser, M., Morin, L., & Bodin, F. (2013). AutoTune: A Plugin-Driven Approach to the Automatic Tuning of Parallel Applications. In P. Manninen & P. Öster (Eds.), *Applied Parallel and Scientific Computing. PARA 2012. Lecture Notes in Computer Science*. Springer. https://doi.org/10.1007/978-3-642-36803-5_24

Miles, M. B., & Huberman, A. M. (1994). *Qualitative data analysis*. Sage.

Miliszewska, I., & Tan, G. (2007). Befriending computer programming: A proposed approach to teaching introductory programming. In E. B. Cohen (Ed.), *Information and Beyond: Part 1. Issues in Informing Science and Information Technology* (Vol. 4, pp. 277-289). Informing Science.

Moons, J., & De Backer, C. (2013). The design and pilot evaluation of an interactive learning environment for introductory programming influenced by cognitive load theory and constructivism. *Computers & Education, 60*(1), 368-384. https://doi.org/10.1016/j.compedu.2012.08.009

Moreno, A., Myller, N., Sutinen, E., & Ben-Ari, M. (2004). Visualizing programs with Jeliot 3. In *Proceedings of the International Working Conference on Advanced Visual Interfaces* (pp. 373-376). ACM. https://doi.org/10.1145/989863.989928

Murata, M., & Kakeshita, T. (2016). Analysis Method of Student Achievement Level Utilizing Web-Based Programming Education Support Tool Pgtacer. In

Proceedings of the 2016 5th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI) (pp. 316-321). IEEE.

Newell, W. H. (1992). Academic disciplines and undergraduate interdisciplinary education: Lessons from the School of Interdisciplinary Studies at Miami University, Ohio. *European Journal of Education, 27*(3), 211-221. https://doi.org/10.2307/1503450

Newell, W. H. (2007). The role of interdisciplinary studies in the liberal arts. *Liberal Arts Online, 7*(1).

Nicolescu, B. (1999, April 19-23). *The transdisciplinary evolution of learning* [Conference presentation]. Symposium on Overcoming the Underdevelopment of Learning at the Annual Meeting of the American Educational Research Association, Montreal, Canada.

Oliveira Aureliano, V. C. (2013). A methodology for teaching programming for beginners. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* (pp. 169-170). ACM.

Opdenakker, R. (2006). Advantages and disadvantages of four interview techniques in qualitative research. *Forum Qualitative Sozialforschung/Forum: Qualitative Social Research, 7*(4), Article 11. http://dx.doi.org/10.17169/fqs-7.4.175

Özgen, K., & Pesen, C. (2008). Probleme dayalı öğrenme yaklaşımı ve öğrencilerin matematiğe yönelik tutumları [Problem-Based Learning Approach and Students' Attitudes towards to Mathematics]. *Dicle Üniversitesi Ziya Gökalp Eğitim Fakültesi Dergisi, 11*, 69-83. https://dergipark.org.tr/en/pub/zgefd/issue/47957/606777

Özmen, B., & Altun, A. (2014). Undergraduate students' experiences in programming: difficulties and obstacles. *Turkish Online Journal of Qualitative Inquiry, 5*(3), 1-27. https://dergipark.org.tr/en/download/article-file/199845

Park, J. Y., & Mills, K. A. (2014). Enhancing interdisciplinary learning with a learning management system. *Journal of Online Learning and Teaching, 10*(2), 299-313. https://jolt.merlot.org/vol10no2/park_0614.pdf

Patton, M. Q. (2014). *Nitel araştırma and değerlendirme yöntemleri* [Qualitative evaluation and research methods] (M. Bütün & S. B. Demir, Trans.). Pegem.

Peppler, K., & Kafai, Y. (2007). What video game making can teach us about learning and literacy: Alternative pathways into participatory cultures. In *Proceedings of the Digital Games Research Association Conference* (pp. 369-376). DiGRA.

Pokress, S. C., & Veiga, J. J. D. (2013). *MIT App Inventor: Enabling personal mobile computing*. arXiv preprint. https://arxiv.org/abs/1310.2830v2

Polaki, M. V. (2002). Using instruction to identify the key features of Basotho elementary students' growth in probabilistic thinking. *Mathematical Thinking and Learning, 4*(4), 285-313. https://doi.org/10.1207/S15327833MTL0404_01

Polit, D. F., & Beck, C. T. (2014). *Essentials of nursing research: Appraising evidence for nursing practice.* Wolters Kluwer Health.

Porter, L., Guzdial, M., McDowell, C., & Simon, B. (2013). Success in introductory programming: What works? *Communications of the ACM, 56*(8), 34-36. https://doi.org/10.1145/2492007.2492020

Pratt, D. (2000). Making sense of the total of two dice. *Journal for Research in Mathematics Education, 31*(5), 602-625. https://doi.org/10.2307/749889

Pruski, L., & Friedman, J. (2014). An integrative approach to teaching mathematics, computer science, and Physics with Matlab. *Mathematics and Computer Education, 48*(1), 6-18.

Rajala, T., Laakso, M., Kaila, E., & Salakoski, T. (2008). Effectiveness of program visualization: A case study with the ViLLE tool. *Journal of Information Technology Education: Innovations in Practice, 7*, 15-32. https://doi.org/10.28945/195

Razzouk, R., & Shute, V. (2012). What is design thinking and why is it important? *Review of Educational Research, 82*(3), 330-348. https://doi.org/10.3102%2F0034654312457429

Repko, A. F., & Szostak, R. (2020). *Interdisciplinary Research: Process and theory* (4th ed.). Sage.

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 Everyone. *Communications of the ACM, 52*(11), 60-67. http://web.media.mit.edu/~mres/scratch/scratch-cacm.pdf

Resnick, M., & Siegel, D. (2015, November 10). *A Different Approach to Coding: How kids are making and remaking themselves from Scratch.* International Open Magazine. https://brightthemag.com/a-different-approach-to-coding-d679b06d83a

Rich, K. M., & Yadav, A. (2019). Infusing computational thinking instruction into elementary mathematics and science: Patterns of teacher implementation. In K. Graziano (Ed.), *Proceedings of the Society for Information Technology & Teacher Education International Conference 2019* (pp. 330-334). AACE. https://www.learntechlib.org/primary/p/207661/

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education, 13*(2), 137-172. https://doi.org/10.1076/csed.13.2.137.14200

Rodriguez, B., Kennicutt, S., Rader, C., & Camp, T. (2017). Assessing Computational Thinking in CS Unplugged Activities. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 501-506). ACM. https://doi.org/10.1145/3017680.3017779

Rodríguez-Martínez, J. A., González-Calero, J. A., & Sáez-López, J. M. (2020). Computational thinking and mathematics using Scratch: an experiment with sixth-grade students. *Interactive Learning Environments, 28*(3), 316-327. https://doi.org/10.1080/10494820.2019.1612448

Saeli, M., Perrenet, J., Jochems, W. M., & Zwaneveld, B. (2011). Teaching programming in secondary school: a pedagogical content knowledge perspective. *Informatics in Education, 10*(1), 73-88. https://www.ceeol.com/search/article-detail?id=69618

Sáez-López, J. M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. *Computers & Education, 97*, 129-141. https://doi.org/10.1016/j.compedu.2016.03.003

Sands, P., Yadav, A., & Good, J. (2018). Computational thinking in K-12: In-service teacher perceptions of computational thinking. In M. S. Khine (Ed.), *Computational thinking in the STEM disciplines* (pp. 151-164). Springer.

Seehorn, D. W., Stephenson, C., Pirmann, T. R., & Powers, K. (2013). CSTA CS K-12 instructional standards and CS curriculum. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education* (pp. 746-746). ACM.

Selby, C., & Woollard, J. (2013). *Computational thinking: the developing definition*. https://eprints.soton.ac.uk/356481

Seng, W. Y., & Yatim, M. H. M. (2014). Computer game as learning and teaching tool for object oriented programming in higher education institution. *Procedia - Social and Behavioral Sciences, 123*, 215-224. https://doi.org/10.1016/j.sbspro.2014.01.1417

Sengupta, P., Kinnebrew, J. S., Basu, S., Biswas, G., & Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: a theoretical framework. *Education and Information Technologies, 18*, 351-380. https://doi.org/10.1007/s10639-012-9240-x

Shute, V. J., Masduki, I., & Donmez, O. (2010). Conceptual framework for modeling, assessing, and supporting competencies within game environments. *Technology, Instruction, Cognition, and Learning, 8*(2), 137-

161. https://www.oldcitypublishing.com/journals/ticl-home/ticl-issue-contents/ticl-volume-8-number-2-2010/ticl-8-2-p-137-161/

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review, 22*, 142-158. https://doi.org/10.1016/j.edurev.2017.09.003

Sins, P. H., Savelsbergh, E. R., & van Joolingen, W. R. (2005). The difficult process of scientific modelling: An analysis of novices reasoning during computer-based modelling. *International Journal of Science Education, 27*(14), 1695-1721. https://doi.org/10.1080/09500690500206408

Smith, P., Smith, J. U., Powlson, D. S., McGill, W. B., Arah, J. R. M., Chertov, O. G., ... & Whitmore, A. P. (1997). A comparison of the performance of nine soil organic matter models using datasets from seven long-term experiments. *Geoderma*, *81*(1-2), 153-225.

Stozhko, N., Bortnik, B., Mironova, L., Tchernysheva, A., & Podshivalova, E. (2015). Interdisciplinary project-based learning: technology for improving student cognition. *Research in Learning Technology*, *23*, Article 27577. https://doi.org/10.3402/rlt.v23.27577

Strauss, A., & Corbin, J. (1990). *Basics of qualitative research* (Vol. 15). Sage.

Swacha, J., & Baszuro, P. (2013). Gamification-based e-learning platform for computer programming education. In N. Reynolds & M. Webb (Eds.), *X World Conference on Computers in Education* (pp. 122-130). WCCE.

Taguchi, N. (2018). Description and explanation of pragmatic development: Quantitative, qualitative, and mixed methods research. *System, 75*, 23-32. https://doi.org/10.1016/j.system.2018.03.010

Takacs, A., Eigner, G., Kovács, L., Rudas, I. J., & Haidegger, T. (2016). Teacher's Kit: Development, Usability, and Communities of Modular Robotic Kits for Classroom Education. *IEEE Robotics & Automation Magazine*, *23*(2), 30-39. https://doi.org/10.1109/MRA.2016.2548754

Tan, J., Guo, X., & Zheng, W. (2014). Case-based teaching using the Laboratory Animal System for learning C/C++ programming. *Computers & Education, 77*, 39-49. https://doi.org/10.1016/j.compedu.2014.04.003

Tashakkori, A., & Teddlie, C. (1998). *Mixed Methodology: Combining Qualitative and Quantitative Approaches*. Sage.

Taub, R., Armoni, M., & Ben-Ari, M. (2012). CS unplugged and middle-school students' views, attitudes, and intentions regarding CS. *ACM Transactions on Computing Education (TOCE), 12*(2), Article 8. https://doi.org/10.1145/2160547.2160551

Teddlie, C., & Tashakkori, A. (2003). Major issues and controversies in the use of mixed methods in the social and behavioral sciences. In A. Tashakkori & C.

Teddlie (Eds.), *Handbook of Mixed Methods in Social and Behavioral Research* (pp. 3-50). Sage.

Thies, R., & Vahrenhold, J. (2013). On plugging unplugged into CS classes. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education* (pp. 365-370). ACM.

Tsai, M.-J., Wang, C.-Y., & Hsu, P.-F. (2019). Developing the computer programming self-efficacy scale for computer literacy education. *Journal of Educational Computing Research, 56*(8), 1345-1360. https://doi.org/10.1177%2F0735633117746747

Tsan, J., Boyer, K. E., & Lynch, C. F. (2016). How Early Does the CS Gender Gap Emerge?: A Study of Collaborative Problem Solving in 5th Grade Computer Science. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 388-393). ACM.

Uzun, A., & Baltalı, S. (2020). Programlama Öğretiminde Kullanılabilecek Yazılımlara İlişkin Öğretmen Görüşleri [Teachers' Opinions About Software That Can Be Used In Teaching Programming]. *Uludağ Üniversitesi Eğitim Fakültesi Dergisi, 33*(1), 129-156. https://doi.org/10.19171/uefad.561833

Vatansever, Ö., & Baltacı Göktalay, Ş. (2018). How Does Teaching Programming through Scratch Affect Problem-solving Skills of 5th and 6th Grade Middle School Students? *International Journal of Eurasia Social Sciences, 9*(33), 1778-1801. http://www.ijoess.com/DergiTamDetay.aspx?ID=2223&Detay=Ozet

Venkatesh, V., Brown, S. A., & Bala, H. (2013). Bridging the qualitative-quantitative divide: Guidelines for conducting mixed methods research in information systems. *MIS Quarterly, 37*(1), 21-54. https://www.jstor.org/stable/43825936?seq=1

Vieira, C., Penmetcha, M., Magana, A., & Matson, E. (2016). Computational Thinking as a Practice of Representation: A Proposed Learning and Assessment Framework. *Journal of Computational Science Education, 7*(1), 21-30. https://doi.org/10.22369/issn.2153-4136/7/1/3

Ward, L., & Parr, J. M. (2010). Revisiting and reframing use: Implications for the integration of ICT. *Computers & Education, 54*(1), 113-122. https://doi.org/10.1016/j.compedu.2009.07.011

Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012). The fairy performance assessment: measuring computational thinking in middle school. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (pp. 215-220). ACM.

Wiedenbeck, S., & Ramalingam, V. (1999). Novice comprehension of small programs written in the procedural and object-oriented styles. *International Journal of Human-Computer Studies*, *51*(1), 71-87. https://doi.org/10.1006/ijhc.1999.0269

Wing, J. M. (2006). Computational Thinking. *Communications of The ACM, 49*(3), 33-35. https://doi.org/10.1145/1118178.1118215

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 366*(1881), 3717-3725. https://doi.org/10.1098/rsta.2008.0118

Winslow, L. E. (1996). Programming pedagogy—a psychological overview. *ACM SIGCSE Bulletin*, *28*(3), 17-22. https://doi.org/10.1145/234867.234872

Wolber, D. (2011). App inventor and real-world motivation. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE)* (Vol. 11, pp. 601-606). ACM. https://doi.org/10.1145/1953163.1953329

Yadav, A., Gretter, S., Good, J., & McLean, T. (2017). Computational thinking in teacher education. In P. J. Rich & C. B. Hodges (Eds.), *Emerging research, practice, and policy on computational thinking* (pp. 205-220). Springer.

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. T. (2014). Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education (TOCE), 14*(1), Article 5. https://doi.org/10.1145/2576872

Yıldırım, A., & Şimşek, H. (2008). *Sosyal bilimlerde nitel araştırma yöntemleri* [Qualitative research methods in the social sciences]. Seçkin.

Yin, R. K. (2009). *Case Study Research: Design and Methods* (4th ed.). Sage.

Yolcu, V. (2018). *Programlama Eğitiminde Robotik Kullanımının Akademik Başarı, Bilgi-İşlemsel Düşünme Becerisi Ve Öğrenme Transferine Etkisi* [The effect of using robotics on academic success, computational thinking skills and transfer of learning in programming education] [Master's thesis]. Süleyman Demirel University, Turkey.

Zhang, J. (2017). Psychological Predicament of Freshmen in the Course of Computer Programming and Its Countermeasures. In E. McAnally, Y. Zhang, R. Green, & I. Tretyakova (Eds.), *Proceedings of the 2nd International Conference on Contemporary Education, Social Sciences and Humanities (ICCESSH 2017)* (pp. 307-309). Atlantis. https://dx.doi.org/10.2991/iccessh-17.2017.72

APPENDICES

A. Activity Plan

<table>
<tr><th>Week</th><th>Topic</th><th>Acquition (Control Group)</th><th>Acquition (Experimental Group)</th></tr>
<tr><td>1</td><td>Applying pretests
CPSES
CTSSES
MAT</td><td></td><td></td></tr>
<tr><td>2</td><td>Lesson Plan 1: What is Small Basic?</td><td colspan="2">Programming achievements:
• Understanding what programming means.
• Establishing relationships between programming languages and spoken languages.
• Recognizing the Small Basic environment and its features.</td></tr>
<tr><td>3</td><td>Lesson Plan 2: Basic commands in Small Basic</td><td colspan="2">Programming achievements:
• Understanding usage of commands:
o TextWindow
o Write
o WriteLine</td></tr>
<tr><td>4</td><td>Lesson Plan 3: Variables</td><td>Programming achievements:
Understanding the concept of variables and what it stands for.
Understanding that variables must have values,
Being aware of the importance of giving a name to a variable without using spaces, special characters, or Turkish characters.</td><td>Programming achievements:
Understanding the concept of variables and what it stands for.
Understanding that variables must have values,
Being aware of the importance of giving a name to a variable without using spaces, special characters, or Turkish characters.
Mathematics achievements:
Recognizing the problem and setting an equation with an unknown for a real-life situation.
Solving equations with an unknown.
Solving problems that require an equation with an unknown.</td></tr>
</table>

5	**Lesson Plan 4:** If-Else	**Programming achievements:** • Applying the concept of variables. • Establishing relationships between variables and operations. • Understanding multiple usage of variables. • Understanding the need for using "If-Else" concept. • Using "If-Else" as a command.	**Programming achievements:** • Applying the concept of variables. • Establishing relationships between variables and operations. • Understanding multiple usage of variables. • Understanding the need for using "If-Else" concept. • Using "If-Else" as a command. **Mathematics achievements: (same as previous week)** • Recognizing the problem and setting an equation with an unknown for a real-life situation. • Solving equations with an unknown. Solving problems that require an equation with an unknown.
6	**Lesson Plan 5:** For Loops	**Programming achievements:** • Remembering the reasons for using loops. • Remembering program lines are shortened with loops. • Understanding the difference between "while" and "for loop."	**Programming achievements:** • Remembering the reasons for using loops. • Remembering program lines are shortened with loops. • Understanding the difference between "while" and "for loop." **Mathematics achievements:** • Expressing rule of number patterns by letter. Finding the questioned term of a pattern whose rule is expressed by a letter.
7	**Lesson Plan 6:** While Loop	**Programming achievements:** • Understanding the difference between "while" and "for loop."	**Programming achievements:** • Understanding the difference between "while" and "for loop." **Mathematics achievements:** • Calculating and interpreting the arithmetic mean of a data group. Finding and interpreting the average, median, and peak value of a data group.

8	**Applying posttests** • CPSES • CTSSES • MAT Scale		
9	**Applying pretest** BASIC Programming Achievement Test		
10	**Interviews with Students**		
After two months	**Applying posttest** Small Basic Programming Achievement Test		

B. Lesson Plans

Week 1 for Both Groups

1. Hafta 1. Ders Planı - Small Basic Nedir.docx

SMALL BASIC İLE PROGRAMLAMA
1.1 Small Basic Ortamı ve İlk Program
Dersin Amacı: Bu derste öğrencilerin Small Basic programının ortamını tanımaları ve "Merhaba Dünya" ile ilk programlarını yazmaları amaçlanmıştır.
Projenin Süresi: 1 ders saati (40 dakika)
Bütünleşik Ders: Türkçe
Hedef Kitle: 7. sınıflar

Kullanılacak Yazılım ve Materyaller: Small Basic

Kazanımlar:

- Programlamanın ne anlama geldiğini kavrar,
- Programlama dilleri ile dünya dilleri arasında ilişki kurar,
- Small Basic ortamını ve araçlarını tanır,
- TextWindow ile program yazabilir.

İçerik & Senaryo:

Konu	**Süre**
● Öğrencilerle "**Hafta 1 – Small Basic**" sunumu paylaşılır. Konuya giriş yapmadan önce Algoritma ve Programlama kavramları hatırlatılır.	5 dakika
● Öğretmen programlama ile dünya dilleri arasında ilişki kurar. o Örn: İnsanları dilleri konuşup, anlaşarak ve iletişim sağlamak için kullanır. Tıpkı insanların birbiri ile anlaşması gibi, makinaları anlamamız ve onların bizi anlamalarını sağlamamız için programlama dilleri kullanırız.	5 dakika
● Small Basic Ortamının Tanıtılması: o **Small Basic ortamı tanıtımı** kaynağından yararlanılır.	15 dakika
● İlk programın yazılması: "Merhaba Dünya" o TextWindow.WriteLine("Merhaba Dünya") komutu yazılırken TextWindow siyah ekranı açmak için kullanılan komut, o WriteLine program çıktısını aldıktan sonra alt satıra geçmek için, Write ise alt satıra geçmeden programı bitirmek için kullanılan metinsel girdi ve çıktılarının olduğu belirtilir. ● Programı kaydetme yöntemi gösterilir.	5 dakika
● Öğrencilerin uygulaması o Hello world / Merhaba dünya yazmaları istenir. o Alt alta faklı renklerde Merhaba dünya yazısını yazabilirler. Bunun için TextWindow.**ForegroundColor** =	8 dakika

"Green" komutunun kullanıldığı belirtilir.	
● Dersin tekrarı	2 dakika

Önemli Vurgulamalar:

- Write veya WriteLine komutlarında ekranda görmek istediklerimizi tırnak ve parantez içine yazıyoruz. **Nokta, parantezler ve tırnak işaretlerinin** tümü, bilgisayarın niyetimizi anlaması için, ifadede doğru yerlere yerleştirilmesi gereken noktalama işaretleri.
- Programımız siyah pencere üzerinde çalıştı bunun nedeni programımızda **nesne** olarak adlandırdığımız **TextWindow** (bazen Konsol olarak adlandırılır) kullanmamız.
- **WriteLine** program çıktısını aldıktan sonra alt satıra geçmek için, **Write** ise alt satıra geçmeden programı bitirmek için kullanılan metinsel girdi ve çıktılarının olduğu tekrar hatırlatılır.

Öneriler:

- "Merhaba Dünya" komutu yazılırken farklı renklerde yazma özelliği kullanılabilir.
    - TextWindow.ForegroundColor = "Green"
    - TextWindow.WriteLine("Merhaba Dünya")

Week 2 for Both Groups

2. Hafta Ders Planı - Small Basic Komutları

SMALL BASIC İLE PROGRAMLAMA
2.1 Small Basic: Değişkenler
Dersin Amacı: Bu derste öğrencilerin Small Basic programını kullanarak nesnel programlamada değişken kavramının önemini öğrenmeleri ve değişken kullanmanın amacını öğrenmeleri amaçlanmıştır.
Projenin Süresi: 1 ders saati (40 dakika)

Bütünleşik Ders:

Hedef Kitle: 7. sınıflar

Kullanılacak Yazılım ve Materyaller: Small Basic

Kazanımlar:

- Değişkenlik kavramını ve ne işe yaradığını öğrenir,
- Değişkenin bir değerleri olması gerektiğinin farkına varır,
- Değişken tanımlarken büyük-küçük harf ayrımı olduğunu fark eder,
- Değişken tanımlarken boşluk, özel karakter ve Türkçe karakter kullanılmaması gerektiğini bilir.

İçerik & Senaryo:

Konu	**Süre**
● Öğrencilere geçen ders öğrendikleri sorulur ve **Write** ile **WriteLine** arasındaki fark hatırlatılır.	3 dakika
● Değişken kavramı öğrencilere kısa bir drama ile öğretilir. Öğretmen Bilişim teknolojileri dersi için her hafta bir "Yardımcı Öğretmen / Co-Teacher" seçeceğin belirtir ve seçeceği kişiyi o ders için "yardımcı" veya "helper" olarak adlandıracağını söyler. Dersten bir öğrenciyi o dersin yardımcısı olarak seçer ve "Bu dersin yardımcı öğrencisi ….. isimli öğrenci" der. ● Tahtaya yardimci = Ali (öğrencinin adı) şeklinde yazar. Bu isimin her hafta değişeceğini belirtir. o değişken kavramında Türkçe ve özel karakterlerin kullanılmadığı hatırlatırlır, o Programlamada kullanılan özel sözcüklerin (if, random, else...vb) değişkenlerde kullanılmayacağı belirtilir, o değişken kavramının rakam ile başlanmayacağı belirtilir, o iki yardimci varsa yardimci1, yardimci2 şeklinde olabileceği söylenir. ● Öğretmen ikinci ders gibi düşünerek tahtada yazan yardimci = Ali isnimi değiştirir ve Ali yerine ikinci öğrencinin adını yazar. Böylelikle değişken kavramının değişilebilir olduğunu, bilgisayarın	5 dakika

anlaması için verileri tutan tanımlama olduğunu belirtir.	
• Small Basic programı açılır ve aşağıdaki program öğrencilere örnek olarak gösterilir: TextWindow.**Write**("İsminizi Girin: ") name = TextWindow.**Read**() TextWindow.**WriteLine**("Merhaba " + name)	7 dakika

• Öğrenci uygulaması:
○ Öğrencilerden aşağıdaki çıktı gibi olacak bir kod yazmaları istenir.
 
 	
○
`TextWindow.Write("İsminizi Girin: ")`
`name = TextWindow.Read()`
`TextWindow.WriteLine("Merhaba " + name)`
 	
• Bu aktiviteden sonra öğrencilerden 3 değişkenli aşağıdaki programı yazmaları istenir. Bunu yaparken metinleri renkli yazmaları da istenir.
 	
 
 	
`TextWindow.ForegroundColor = "yellow"`
	20 dakika

```
TextWindow.Write("İsminizi
Girin: ")
name = TextWindow.Read()
TextWindow.ForegroundColor =
"yellow"
TextWindow.WriteLine("Merhaba "
+ name +"!")

TextWindow.ForegroundColor =
"green"
TextWindow.Write("Yaşınızı
giriniz: ")
age=TextWindow.Read()
TextWindow.ForegroundColor =
"green"
TextWindow.WriteLine(age + "
yaşında olmak çok güzel
olmalı!")

TextWindow.ForegroundColor =
"cyan"
TextWindow.Write("Bugün akşam
yemeğinde ne yemek istersin? ")
dinner=TextWindow.Read()
TextWindow.ForegroundColor =
"cyan"
TextWindow.WriteLine("Harika bir
seçim! " + dinner + " yemeğini
ben de severim.")
```

- Dersin tekrarı

5 dakik

Önemli Vurgulamalar:

- **Write** veya **WriteLine** komutları tekrar vurgulanır.
    - **WriteLine** program çıktısını aldıktan sonra alt satıra geçmek için, **Write** ise alt satıra geçmeden programı bitirmek için kullanılan metinsel girdi ve çıktılarının olduğu tekrar hatırlatılır.
- name = TextWindow.**Read**() kodunda bulunan Read () komutunun bir işlemci olduğu ve temel olarak bilgisayara kullanıcının bir metin girmesini ve ENTER tuşuna basmasını beklemesini söylediği vurgulanır.
- **ForegroundColor** komutu ile yazılarımızı renkli yazabileceğimiz belirtilir. Renkleri ister merinsel olarak **“”** içine, istersek her renk için belirtilen r**enk kodları** ile yazabileceğimiz paylaşılır ve öğrencilere renk kodları paylaşılır. (Google’da HTML Color Chart/Codes şeklinde aratılabilir.)
    - TextWindow.**ForegroundColor** = **"Green"**
    - TextWindow.**ForegroundColor** = **#33691e**

Öneriler:

Renkler:

- Black
- Blue
- Cyan
- Gray
- Green
- Magenta
- Red
- White
- Yellow
- DarkBlue
- DarkCyan
- DarkGray
- DarkGreen
- DarkMagenta
- DarkRed
- DarkYellow

Week 3 for Experimental Group (Mathematics Integration)

3. Hafta Ders Planı (Math)- Small Basic: Değişkenler

<table>
<tr><td colspan="2">SMALL BASIC İLE PROGRAMLAMA</td></tr>
<tr><td colspan="2">3.1 Small Basic: Değişkenler: Matematik</td></tr>
<tr><td colspan="2">Dersin Amacı: Bu derste öğrencilerin Small Basic programını</td></tr>
<tr><td colspan="2">Projenin Süresi: 1 ders saati (40 dakika)</td></tr>
<tr><td colspan="2">Bütünleşik Ders: Matematik</td></tr>
<tr><td colspan="2">Hedef Kitle: 7. sınıflar</td></tr>
<tr><td colspan="2">Kullanılacak Yazılım ve Materyaller: Small Basic</td></tr>
<tr><td colspan="2">Kazanımlar:
Bilişim Teknolojileri:
● Değişkenlik kavramını anlar,
● Değişkenler ve program sırası arasında ilişki kurar,
● Bir programda birden fazla değişken kullanılabileceğini kavrar.
Matematik:
● Problemi tanır ve verilen gerçek hayat durumlarına uygun birinci dereceden bir bilinmeyenli denklem kurar.
● Birinci dereceden bir bilinmeyenli denklemleri çözer.
● Birinci dereceden bir bilinmeyenli denklem kurmayı gerektiren problemleri çözer.</td></tr>
<tr><td colspan="2">İçerik & Senaryo:
<table>
<tr><th>Konu</th><th>Süre</th></tr>
<tr><td>● Öğrencilere geçen ders öğrendikleri sorulur ve değişkenlik
kavramının tanımı harlatılır.</td><td>3 dakika</td></tr>
<tr><td>● Matematik kitabından alınan aşağıdaki problem tahtaya</td><td></td></tr>
</table>
</td></tr>
</table>

yansıtılır (3. Hafta Değişken Matematik Problemi PowerPoint dokümanı):
Ali'nin alacağı 2500 TL değerindeki cep telefonu için
mağaza 2 farklı ödeme seçeneği sunmuştur.
Ali telefonu 12 taksitle alırsa aylık ödemesi 225 TL
olacakken, peşin alırsa satış fiyatı üzerinden 150 TL
indirim kazanacaktır.
Telefonun taksitli ve peşin satış fiyatlarını hesaplayınız.
● Problemin matematiksel çözümü öğrencilere adım adım sorulur ve nakit alımda sonucun 2500-150 TL olduğu, taksitli alımda ise 225*12 TL olduğu paylaşılır.
● Programda **değişken** kavramının ne olduğu öğrencilere sorulur ve ödeme şekline göre değişeceğinden **ödenmesi gereken tutarın** burada değişken olduğu belirtilir.
● Fikirlerin paylaşımından sonra öğrencilerden sonucun ekrana yazılacağı programın yazılması istenir.
 
`TextWindow.WriteLine("Lütfen telefonun fiyatını giriniz: ")`
`butce = TextWindow.Read()`
`pesin= butce-150`
`taksit = 225*12`
	10 dakika

TextWindow.WriteLine ("Bütçenize göre ödeme planınız gereken miktar aşağıdaki gibidir: ") TextWindow.WriteLine ("Peşin ödemede: " +pesin+ " TL ödemeniz gerekmektedir.") TextWindow.WriteLine ("Taksitli ödemede: " +taksit+ " TL ödemeniz gerekmektedir.")	
• Öğrenci uygulaması:	
• Dersin tekrarı • Programdaki mantık hatası öğrencilere sorulur. Girilen bütçe ile mağazanın taban bütçesinin karşılaştırılması gerektiği ve peşin ödenen bütçenin taksitliden küçük olması gerektiği öğrencilerle tartışılır. • Bir sonraki ders için bu konuya çözüm bulmaları istenir. (If-Else)	7 dakika

Week 3 For Control Group (Traditional)

3. Hafta Ders Planı (Traditional)- Small Basic_ Değişkenler

SMALL BASIC İLE PROGRAMLAMA
3. Small Basic: Değişkenler: Klasik Yöntem
Dersin Amacı: Bu derste öğrencilerin Small Basic programını kullanarak değişken kavramını matematik problemleri ile bağlantı kurarak uygular ve öğrenir.
Projenin Süresi: 1 ders saati (40 dakika)
Hedef Kitle: 7. sınıflar

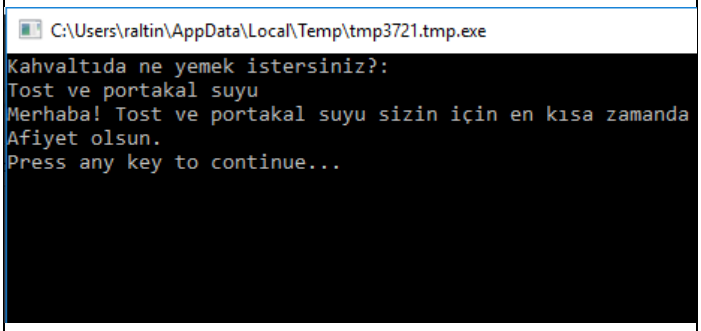
Kullanılacak Yazılım ve Materyaller: Small Basic

Kazanımlar:

- Değişkenlik kavramını anlar,
- Değişkenler ve program sırası arasında ilişki kurar,
- Bir programda birden fazla değişken kullanılabileceğini kavrar.

İçerik & Senaryo:

Konu	**Süre**
● Öğrencilere geçen ders öğrendikleri sorulur ve **değişkenlik** kavramının tanımı harlatılır.	3 dakika
● Öğrencilerle kahvaltıda iki seçeneklerinin olduğu ve bu seçeneğe göre sipariş verebileceklerini belirten aşağıdaki soru paylaşılır. ○ Çok güzel bir Pazar günü olduğunu hayal edelim. Hafta sonu ailenle kahvaltı yapmaya gittin. Servisi yapan görevli sana kahvaltıda ne yemek istediğini sordu. Senden aldığı cevaba göre de kahvaltıda yemek istediğini söylüyor. Buna göre yazacağımız program size kahvaltı seçeneğini sorup sizden aldığı yanıta göre mönüyü yazdırmalı. ● Programda **değişken** kavramının ne olduğu öğrencilere sorulur ve alınan cevaba göre değişeceği için **kahvaltıda istenilenin** değişken olduğu belirtilir. ● Öğrencilere programın nasıl olması gerektiği sorulur ve gelen yanıtlar tahtaya yazılır. Örnek olarak aşağıdaki program çıktısı sunulur ve	10 dakika

öğrencilerden kendi programlarını yazmaları istenir.
 
 	
`TextWindow.WriteLine("Kahvaltıda ne yemek istersiniz?: ")`
`kahvalti = TextWindow.Read()`
 	
`TextWindow.WriteLine ("Merhaba! " +kahvalti+ " sizin için en kısa zamanda hazır olacak.")`
`TextWindow.WriteLine ("Afiyet olsun.")`	

• Öğrenci uygulaması:	20 dakika
• Dersin tekrarı • Öğrencilerde bir sonraki ders için iki seçenek sunacakları ve bu seçeneklere göre menü çıkaran bir program yazmaları için ne yapılması gerektiği ile ilgili düşünmeleri istenir. (If kullanımı)	7 dakika

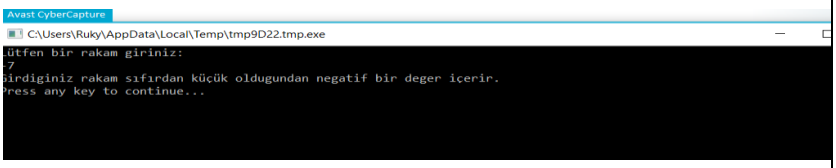
Week 4 for Experimental Group (Mathematics Integration)

4. Hafta Ders Planı (Math)- Small Basic: If Else

SMALL BASIC İLE PROGRAMLAMA
4.1 Small Basic: If-Else: Matematik
Dersin Amacı: Bu derste öğrencilerin Small Basic programını kullanarak
Projenin Süresi: 1 ders saati (40 dakika)
Bütünleşik Ders: Matematik
Hedef Kitle: 7. sınıflar
Kullanılacak Yazılım ve Materyaller: Small Basic
Kazanımlar: **Bilişim Teknolojileri:** ● Değişkenlik kavramını matematik problemleri ile bütünleştirerek uygular, ● Değişkenler ve işlemler arasında ilişki kurar, ● Bir programda birden fazla değişken kullanılabileceğini kavrar, ● If-Else kavramının neden kullandığını öğrenir **Matematik:** ● Problemi tanır ve verilen gerçek hayat durumlarına uygun birinci dereceden bir bilinmeyenli denklem kurar. ● Birinci dereceden bir bilinmeyenli denklemleri çözer. ● Birinci dereceden bir bilinmeyenli denklem kurmayı gerektiren problemleri çözer.
İçerik & Senaryo:

Konu	**Süre**
● Öğrencilere geçen ders öğrendikleri sorulur ve **son yapılan probleme** yönelik hatırlatma yapılır.	3 dakika
● Öğrencilere telefon alımı ile ilgili bir önceki ders uygulanan aşağıdaki matematik problemi hatırlatılır	

○ Ali'nin alacağı 2500 TL değerindeki cep telefonu için mağaza 2 farklı ödeme seçeneği sunmuştur. Ali telefonu 12 taksitle alırsa aylık ödemesi 225 TL olacakken, peşin alırsa satış fiyatı üzerinden 150 TL indirim kazanacaktır. Telefonun taksitli ve peşin satış fiyatlarını hesaplayınız. ve aşağıdaki sorular yönlendirilir: ○ Taksit ve nakit alımı arasındaki farklar nelerdir? ○ Alınan ürünün toplam fiyatına bakıldığında taksitli ödemeye mi yoksa nakit ödemeye mi daha fazla bütçe ayırmamız gerekir? ○ Bu programda mağaza sahibinin telefonun satacağı en düşük fiyatın nakit fiyat yani 2350 TL olduğunu kabul edelim. Bütçemiz 2350 TL'den az ise program tarafından nasıl bir yanıt almalıyız? ● Bu sorulardan sonra öğrencilere algoritmada kullandıklar If-Else kavamı hatırlatılır ve Small Basic'de kullanımı aşağıdaki program ile gösterilir:	15 dakika




```
TextWindow.WriteLine("Lütfen bir rakam
giriniz: ")
rakam = TextWindow.Read()

If (rakam<0) then
TextWindow.WriteLine ("Girdiğiniz rakam
sıfırdan küçük olduğundan negatif bir
değer içerir.")
Else
TextWindow.WriteLine ("Girdiginiz rakam
sıfırdan büyük olduğundan pozitif bir
değer içerir.")
EndIf
```

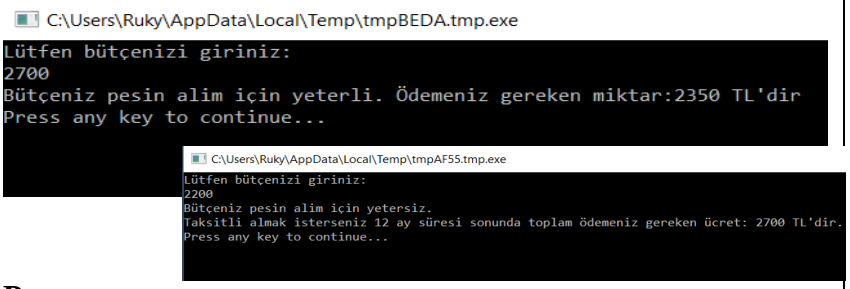
- Bu programdan sonra bir de aşağıdaki program ile iç içe If (NestedIf) kavramı gösterilir:

```
TextWindow.WriteLine("Lütfen karşılaştırmak
istediğiniz birinci rakamı giriniz: ")
```

```
rakam1 = TextWindow.Read()

TextWindow.WriteLine("Lütfen karşılaştırmak
istediğiniz ikinci rakamı giriniz: ")
rakam2 = TextWindow.Read()

TextWindow.WriteLine("Lütfen karşılaştırmak
istediğiniz üçüncü rakamı giriniz: ")
rakam3 = TextWindow.Read()

If (rakam1>rakam2 And rakam1>rakam3) then
  TextWindow.WriteLine (rakam1+ " en
büyüktür.")
Elseif (rakam2>rakam3) then
  TextWindow.WriteLine (rakam2+ " en
büyüktür.")
Else
   TextWindow.WriteLine (rakam3+ " en
büyüktür.")
EndIf
```

- Öğrenci uygulaması: Bir önceki ders programlanan matematik programına if kavramı eklenir ve eğer müşterinin girdiği bütçe peşin ve taksitli alım fiyatından küçükse programın bütçeniz yetersiz mesajı vermesi istenir.

17 dakika



Program:

```
TextWindow.WriteLine("Lütfen bütçenizi
giriniz: ")
butce = TextWindow.Read()

pesin= 2500-150
taksit = 225*12

If(butce > pesin) Then
```

```
  TextWindow.WriteLine ("Bütçeniz peşin
alım için yeterli. Ödemeniz gereken
miktar:" +pesin+ " TL'dir" )
Else
  TextWindow.WriteLine ("Bütçeniz peşin
alım için yetersiz.")
     TextWindow.WriteLine ("Taksitli
almak isterseniz 12 ay süresi sonunda
toplam ödemeniz gereken ücret: "
+taksit+" TL'dir.")

EndIf
```

- Dersin tekrarı

Week 4 For Control Group (Traditional)

4. Hafta Ders Planı (Traditional)-If Else

SMALL BASIC İLE PROGRAMLAMA
4. Small Basic: If – Else: Klasik Yöntem
Dersin Amacı: Bu derste öğrencilerin Small Basic programını kullanarak değişken kavramı ile birlikte karşılaştırma için kullanılan if-else kavramını uygulamak ve öğrenmek.
Projenin Süresi: 1 ders saati (40 dakika)
Hedef Kitle: 7. sınıflar
Kullanılacak Yazılım ve Materyaller: Small Basic
Kazanımlar: ● Değişkenlik kavramını matematik problemleri ile bütünleştirerek uygular, ● Değişkenler ve işlemler arasında ilişki kurar, ● Bir programda birden fazla değişken kullanılabileceğini kavrar, ● If-Else kavramının neden kullandığını öğrenir

İçerik & Senaryo:

<table>
<tr><th>Konu</th><th>Süre</th></tr>
<tr><td>

- Öğrencilere geçen ders öğrendikleri sorulur ve **son yapılan probleme** yönelik hatırlatma yapılır.

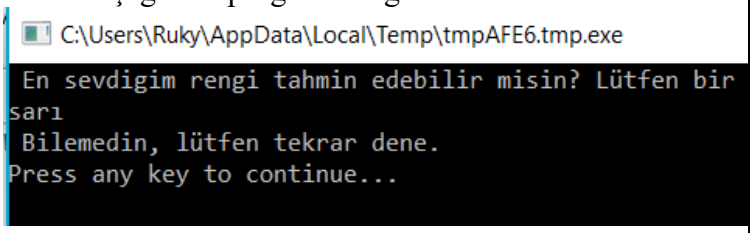
</td><td>3 dakika</td></tr>
<tr><td>

- Öğrencilere kahvaltı seçeneği ile igili bir önceki ders uygulanan aşağıdaki program hatırlatılır
  - Hafta sonu ailenle gideceğin kahvaltı için iki seçenekten oluşan bir program yazınız. Program size kahvaltı seçeneğini sorup sizden aldığı yanıta göre mönüyü yazdırmalı.

ve aşağıdaki soru yönlendirilir:

  - Kahvaltıda bizlere iki seçenek sunulsa ve biz o seçeneklerden birini seçsek nasıl bir program yazabiliriz?

- Bu soru ile öğrencilerle beyin fırtınası yaptıktan sonra öğrencilere algoritmada kullandıklar If-Else kavramı hatırlatılır ve Small Basic'de kullanımı aşağıdaki program ile gösterilir:

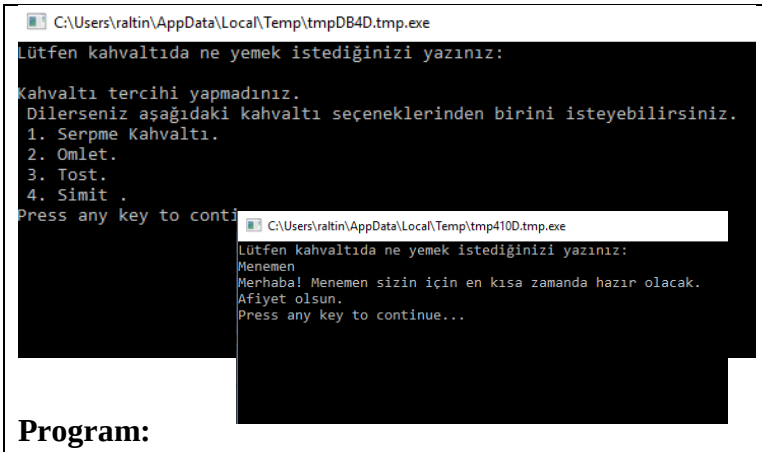



```
TextWindow.WriteLine("En sevdiğim
rengi tahmin edebilir misin? Lütfen
bir renk giriniz: ")
renk = TextWindow.Read()

If (renk="kırmızı") then
  TextWindow.WriteLine ("Doğru
tahmin! En sevdiğim renk kırmızı.")
Else
  TextWindow.WriteLine ("Bilemedin,
lütfen tekrar dene.")
EndIf
```

</td><td>15 dakika</td></tr>
</table>

• Bu programdan sonra bir de aşağıdaki program ile iç içe If (NestedIf) kavramı gösterilir:
`TextWindow.WriteLine("Lütfen karşılaştırmak istediğiniz birinci rakamı giriniz: ")`
`rakam1 = TextWindow.Read()`
 	
`TextWindow.WriteLine("Lütfen karşılaştırmak istediğiniz ikinci rakamı giriniz: ")`
`rakam2 = TextWindow.Read()`
 	
`TextWindow.WriteLine("Lütfen karşılaştırmak istediğiniz üçüncü rakamı giriniz: ")`
`rakam3 = TextWindow.Read()`
 	
`If (rakam1>rakam2 And rakam1>rakam3) then`
`  TextWindow.WriteLine (rakam1+ " en büyüktür.")`
`Elseif (rakam2>rakam3) then`
`  TextWindow.WriteLine (rakam2+ " en büyüktür.")`
`Else`
`  TextWindow.WriteLine (rakam3+ " en büyüktür.")`
`  EndIf`	

• Öğrenci uygulaması: Bir önceki ders programlanan kahvaltı programına if kavramı eklenir ve eğer müşteri herhangi bir tercih yapmazsa hazır çalışanlara danışılabileceğine dair bir mesajım ekrana yazılacağı programı yazmaları istenir	17 dakika



Program:

```
TextWindow.WriteLine("Lütfen kahvaltıda ne yemek istediğinizi yazınız: ")

kahvalti = TextWindow.Read()

If(kahvalti <> "") Then
  TextWindow.WriteLine ("Merhaba! " +kahvalti+ " sizin için en kısa zamanda hazır olacak.")
TextWindow.WriteLine ("Afiyet olsun.")

Else
 TextWindow.WriteLine ("Kahvaltý tercihi yapmadınız.")
 TextWindow.WriteLine (" Dilerseniz aşağıdaki kahvaltı seçeneklerinden birini isteyebilirsiniz.")
 TextWindow.WriteLine (" 1. Serpme Kahvaltı.")
 TextWindow.WriteLine (" 2. Omlet.")
 TextWindow.WriteLine (" 3. Tost.")
 TextWindow.WriteLine (" 4. Simit .")
EndIf
```

- Dersin tekrarı

5 dakika

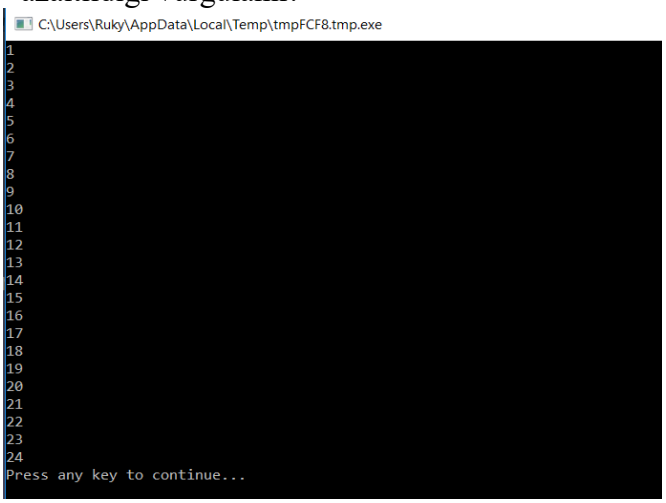
Week 5 for Experimental Group (Mathematics Integration)

5. Hafta Ders Planı - Small Basic (Math): For Loop / For Döngüsü

SMALL BASIC İLE PROGRAMLAMA
5. Small Basic: For Döngüsü: Matematik
Dersin Amacı: Bu derste öğrencilerin Small Basic programını kullanarak **for döngü** kavramı öğrenmek.
Projenin Süresi: 1 ders saati (40 dakika)
Bütünleşik Ders: Matematik
Hedef Kitle: 7. sınıflar
Kullanılacak Yazılım ve Materyaller: Small Basic
Kazanımlar: **Bilişim Teknolojileri:** ● Döngü kavramının neden kullanıldığını öğrenir, ● Döngülerle program satırlarının kısaldığını farkeder, ● **For** döngüsünde değişkenlere başlangıç ve bitiş noktası vererek kullanıldığını kavrar, **Matematik:** • Sayı örüntülerinin kuralını harfle ifade eder, kuralı harfle ifade edilen örüntünün istenilen terimini bulur.

İçerik & Senaryo:

Konu	Süre
● Öğrencilere geçen ders öğrendikleri sorulur ve if-else komutları sözel olarak tekrar edilir.	3 dakika

● Döngü kavramı öğrencilere aşağıdaki videonun izletilmesi ile başlar:
○ Flocabulary: Loops Videosu https://www.flocabulary.com/unit/coding-for-loops/
● Döngülerin bilgisayara tekrarını etmesini istediğimi komutlar için kullanıldığı belirtilir. Bunun için 1'den 24'e kadar rakamların ekrana gelmesini sağlayan bir program yazmaları gerektiğini ve hangi yollarla yazılabileceği öğrencilere sorulur.
● Alınan cevaplardan sonra aynı program çıktısını veren aşağıdaki iki program öğrencilere gösterilir ve For döngüsü kullanımı ile satır sayısının ne kadar azaltıldığı vurgulanır.
 
 	
Program 1:
`TextWindow.WriteLine("1")`
`TextWindow.WriteLine("2")`
`TextWindow.WriteLine("3")`
`TextWindow.WriteLine("4")`
`TextWindow.WriteLine("5")`
`TextWindow.WriteLine("6")`
`TextWindow.WriteLine("7")`
`TextWindow.WriteLine("8")`
`TextWindow.WriteLine("9")`
	15 dakika

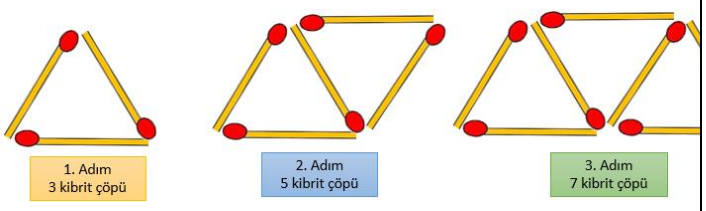
```
TextWindow.WriteLine("10")
TextWindow.WriteLine("11")
TextWindow.WriteLine("12")
TextWindow.WriteLine("13")
TextWindow.WriteLine("14")
TextWindow.WriteLine("15")
TextWindow.WriteLine("16")
TextWindow.WriteLine("17")
TextWindow.WriteLine("18")
TextWindow.WriteLine("19")
TextWindow.WriteLine("20")
TextWindow.WriteLine("21")
TextWindow.WriteLine("22")
TextWindow.WriteLine("23")
TextWindow.WriteLine("24")
```

Program 2:

```
For i = 1 To 24
   TextWindow.WriteLine(i)
EndFor
```

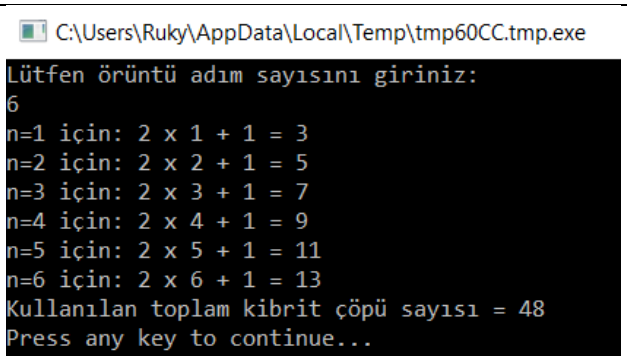
- Bu programda kullanılan **i** değerinin 1'den 24'e kadar gelen sayıları tutması için atanan değişken olduğu vurgulanır. **For** döngüsünde döngünün **başlangıç** ve **bitiş** değeri olduğu vurgulanır ve pro□ramın bu değerler arasında tekrar ettiği belirtilir.

Aşağıdaki matematik probleminin öğrenciler tarafından programlanması istenir:



Yukarıda kibrit çöpleri kullanılarak oluşturulmuş olan bir şekil örüntüsünün ilk üç adımı verilmiştir. Bu örüntünün ilk 6 adımında kullanılan toplam kibrit çöpü sayısı kaçtır?

17 dakika




```
TextWindow.WriteLine("Lütfen örüntü adım sayısını giriniz: ")
sayi = TextWindow.Read()

For i = 1 to sayi
  formul = 2*i+1
  TextWindow.WriteLine( "n=" +i+ " için: 2 x " + i + " + 1 = " + formul)
  toplam=toplam+formul
EndFor

TextWindow.WriteLine  ("Kullanılan  toplam kibrit çöpü sayısı = " +toplam)
```

- Dersin tekrarı

Week 5 For Control Group (Traditional)

5. Hafta Ders Planı (Traditional): For Loop / For Döngüsü

SMALL BASIC İLE PROGRAMLAMA
5. Small Basic: Döngüler – For Döngüsü: Klasik Yöntem
Dersin Amacı: Bu derste öğrencilerin Small Basic programını kullanarak **for döngü** kavramı öğrenmek.
Projenin Süresi: 1 ders saati (40 dakika)

Hedef Kitle: 7. sınıflar

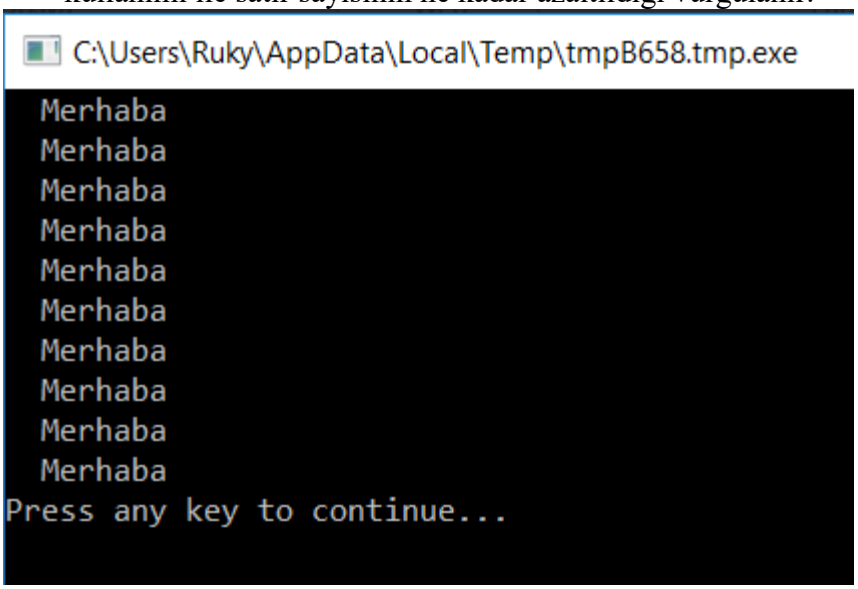
Kullanılacak Yazılım ve Materyaller: Small Basic

Kazanımlar:

- Döngü kavramının neden kullanıldığını öğrenir,
- Döngülerle program satırlarının kısaldığını farkeder,
- **For** döngüsünde değişkenlere başlangıç ve bitiş noktası vererek kullanıldığını kavrar,

İçerik & Senaryo:

Konu	Süre
● Öğrencilere geçen ders öğrendikleri sorulur ve if-else komutları sözel olarak tekrar edilir.	3 dakika
● Döngü kavramı öğrencilere aşağıdaki videonun izletilmesi ile başlar: ○ **Flocabulary: Loops Videosu** ● Döngülerin bilgisayara tekrarını etmesini istediğimi komutlar için kullanıldığı belirtilir. Ekrana 10 kere "Merhaba" ekrana gelmesini sağlayan bir program yazmaları gerektiğini ve hangi yollarla yazılabileceği öğrencilere sorulur. ● Alınan cevaplardan sonra aynı program çıktısını veren aşağıdaki iki program öğrencilere gösterilir ve For döngüsü kullanımı ile satır sayısının ne kadar azaltıldığı vurgulanır.	15 dakika



Program 1:

```
TextWindow.WriteLine("Merhaba")
TextWindow.WriteLine("Merhaba")
TextWindow.WriteLine("Merhaba")
TextWindow.WriteLine("Merhaba")
TextWindow.WriteLine("Merhaba")
TextWindow.WriteLine("Merhaba")
TextWindow.WriteLine("Merhaba")
TextWindow.WriteLine("Merhaba")
TextWindow.WriteLine("Merhaba")
TextWindow.WriteLine("Merhaba")
```

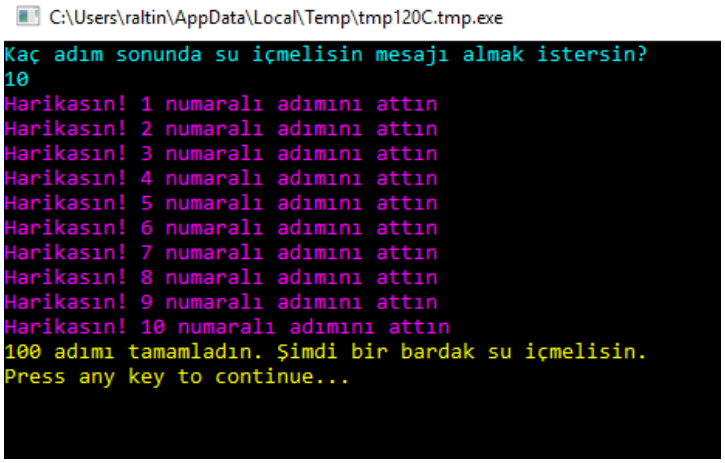
Program 2:

```
For i = 1 To 10
   TextWindow.WriteLine(
TextWindow.WriteLine(“Merhaba”)
EndFor
```

- Bu programda kullanılan **i** değerinin 1’den 10’a kadar gelen sayıları tutması için atanan **değişken** olduğu vurgulanır.

Artık hayatınızda sporun da olmasını istiyorsunuz. Sporun yanında su içmeye de önem gösteriyorsunuz. Öyle bir program yazmalıyız ki Belirlediğin adım sayısına ulaştığında bana “Su İçmelisin” mesajı vermeli.

17 dakika




```
TextWindow.ForegroundColor="cyan"
TextWindow.WriteLine ("Kaç adım sonunda su
içmelisin mesajı almak istersin?")
adim= TextWindow.ReadNumber()

For i = 1 To adim
  TextWindow.ForegroundColor="magenta"
  TextWindow.WriteLine("Harikasın! " +i+ "
numaralı adımını attın")
EndFor
TextWindow.ForegroundColor="yellow"
TextWindow.WriteLine("100 adımı tamamladın.
Şimdi bir bardak su içmelisin.")
```

- Dersin tekrarı

	5 dakika

Week 6 for Experimental Group (Mathematics Integration)

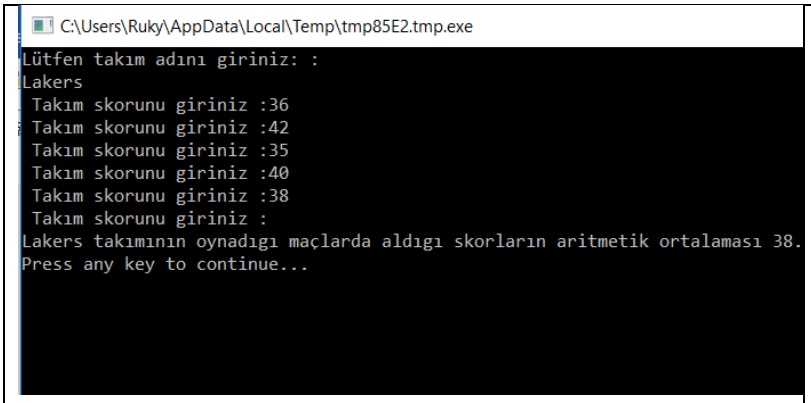
6. Hafta Ders Planı Small Basic (Math): While Loop / While Döngüsü

SMALL BASIC İLE PROGRAMLAMA
6. Small Basic: Döngüler: While Dögüsü: Matematik
Dersin Amacı: Bu derste öğrencilerin Small Basic programını kullanarak **while döngü** kavramı öğrenmek.
Projenin Süresi: 1 ders saati (40 dakika)
Bütünleşik Ders: Matematik
Hedef Kitle: 7. sınıflar
Kullanılacak Yazılım ve Materyaller: Small Basic
Kazanımlar: **Bilişim Teknolojileri:** ● Döngü kavramının neden kullanıldığını öğrenir, ● Döngülerle program satırlarının kısaldığını fark eder, ● Döngülerin değişkenlere başlangıç ve bitiş noktası vererek kullanıldığını kavrar, ● While ve for döngüsü arasındaki farkı öğrenir. **Matematik:** ● Bir veri grubuna ait aritmetik ortalamayı hesaplar ve yorumlar. ● Bir veri grubuna ait ortalama, ortanca ve tepe değeri bulur ve yorumlar.
İçerik & Senaryo:

Konu	Süre
• Öğrencilere geçen ders öğrendikleri sorulur ve for döngüsünü hangi durumlarda kullandığımız sorulur.	3 dakika

• **For** döngüsü belirli sayı aralıkları arasında kullanıldığı öğrencilere bir önceki derste verilen 1'den 24'e kadar sayıların ekrana yazılması örneği ile hatırlatılır.
`For i = 1 To 24`
`  TextWindow.WriteLine(i)`
`EndFor`
• While döngüsünü kullanırken belli bir koşul olması gerektiği söylenir ve aşağıdaki örnek paylaşılır.
`number = 40`
`While (number > 1)`
`  TextWindow.WriteLine(number)`
`  number = number / 2`
`EndWhile`
• Bu programdaki koşul öğrencilere sorulur ve açıklaması şu şekilde yapılır:
Bu döngüde number'a 40 değerini atıyoruz ve gelen sayının ikiye bölümü sayı 1'den büyük olduğu sürece While döngüsünü çalıştırıyoruz.
	15 dakika

Öğretmen 6. Hafta problemini tahtaya yansıtır ve öğrencilere aritmetik ortamalanın nasıl alındığına dair sorular sorar. Aritmetik ortalama hesaplanmasının nasıl yapıldığı hatırlatıldıktan sonra while döngüsü kullanarak bu problemin nasıl yapılabileceği öğrencilere sorulur. **Soru:** Bir basketbol takımı 5 maçta sırasıyla 36, 42, 35, 40, 38 sayı atmıştır. Bu takımın 5 maçtaki sayı ortalamasını hesaplayınız	17 dakika




```
toplam =0
macsayisi=0

TextWindow.WriteLine("Lütfen takım adını
giriniz: :")
takim = TextWindow.Read()

While takim<>""
  TextWindow.Write(" Takım skorunu giriniz
:")
  skor = textwindow.ReadNumber()
  If skor > 0 Then
    toplam= toplam+skor
    macsayisi=macsayisi+1
  Else
    Goto end
  EndIf
EndWhile
end :
   TextWindow.WriteLine(takim+  "  takımının
   oynadığı   maçlarda   aldığı   skorların
   aritmetik ortalaması " +toplam/macsayisi
   + " şeklindedir.")
```

NOT: While döngüsünde belirtilen koşulun ekrana sayı girmeden enter'a basmak olduğu, bu aşamaya kadar döngünün sürekli devam edeceği öğrencilere net bir şekilde ifade edilmeli.

● Dersin tekrarı	5 dakika

Week 6 For Control Group (Traditional)

6. Hafta Ders Planı - Small Basic (Traditional): For While / While Döngüsü

SMALL BASIC İLE PROGRAMLAMA
6. Small Basic: Döngüler: While Dögüsü: Klasik Yöntem
Dersin Amacı: Bu derste öğrencilerin Small Basic programını kullanarak **while döngü** kavramı öğrenmek.
Projenin Süresi: 1 ders saati (40 dakika)
Hedef Kitle: 7. sınıflar
Kullanılacak Yazılım ve Materyaller: Small Basic
Kazanımlar: ● Döngü kavramının neden kullanıldığını öğrenir, ● Döngülerle program satırlarının kısaldığını fark eder, ● Döngülerin değişkenlere başlangıç ve bitiş noktası vererek kullanıldığını kavrar, ● While ve for döngüsü arasındaki farkı öğrenir.
İçerik & Senaryo:

Konu	**Süre**
● Öğrencilere geçen ders öğrendikleri sorulur ve for döngüsünü hangi durumlarda kullandığımız sorulur.	3 dakika
● **For** döngüsü belirli sayı aralıkları arasında kullanıldığı öğrencilere bir önceki derste verilen Ekrana 10 kere "Merhaba" ekrana gelmesini sağlayan program örneği ile hatırlatılır. `For i = 1 To 10`	12 dakika

```
    TextWindow.WriteLine(
TextWindow.WriteLine("Merhaba")
EndFor
```

- While döngüsünü kullanırken belli bir koşul olması gerektiği söylenir ve aşağıdaki örnek paylaşılır.

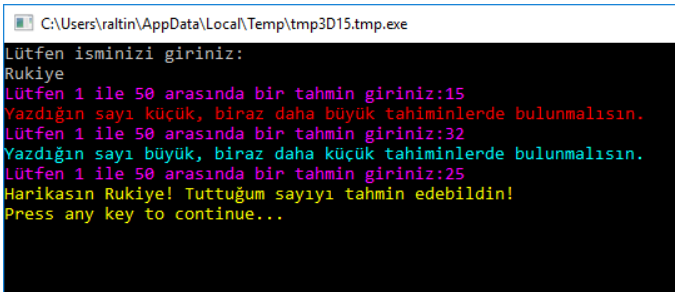
```
number = 40
While (number > 1)
    TextWindow.WriteLine(number)
    number = number / 2
EndWhile
```

- Bu programdaki koşul öğrencilere sorulur ve açıklaması şu şekilde yapılır:

Bu döngüde number'a 40 değerini atıyoruz ve gelen sayının ikiye bölümü sayı 1'den büyük olduğu sürece While döngüsünü çalıştırıyoruz. .

Evde çok sıkılan Pelin bir oyun programlamak ister. Programda tuttuğu sayıyı, oyuncunun tahmin ederek bulmasını sağlayan bir oyun düşünür. Eğer oyuncunun girdiği sayı, programda tutulan sayıdan büyükse "**Yazdığın sayı büyük, biraz daha küçük tahiminlerde bulunmalısın**", küçükse "**Yazdığın sayı küçük, biraz daha büyük tahiminlerde bulunmalısın**" mesajlarının verilen bir program yazar. Sizler de yazabilir misiniz?

20 dakika




```
TextWindow.WriteLine("Lütfen isminizi
giriniz:" )
isim = TextWindow.Read()

sayi = 25
cevap = 0
While (cevap <> sayi)
       TextWindow.ForegroundColor="Magenta"
    TextWindow.Write("Lütfen 1 ile 50
arasında bir tahmin giriniz:")
    cevap = TextWindow.ReadNumber()
    If (cevap = sayi) Then ' Player guessed
correctly
      TextWindow.ForegroundColor="yellow"
```

```
        TextWindow.WriteLine("Harikasın "
+isim+ "! Tuttuğum sayıyı tahmin edebildin!")
    ElseIf (cevap > sayi) Then
      TextWindow.ForegroundColor="cyan"
     TextWindow.WriteLine("Yazdığın sayı
büyük, biraz daha küçük tahiminlerde
bulunmalısın.")
   Else
     TextWindow.ForegroundColor="red"
     TextWindow.WriteLine("Yazdığın sayı
küçük, biraz daha büyük tahiminlerde
bulunmalısın.")
   EndIf
     EndWhile
```

- Dersin tekrarı

C. Visuals for Lessons

Week 1 For Both Groups

HATIRLAYALIM

Algorithma?

Belli bir problemi çözmek veya belirli bir amaca ulaşmak için tasarladığımız ve izlediğimiz yol.

Programlama?

Var olan bir problemi çözmek amacıyla bilgisayar dili kullanılarak oluşturduğumuz anlatımlar.

Bir anlatımı bilgisayar dili ile programlamadan önce algoritma ile izleyeceğimiz yolu tasarlarız.

BU DERSIMIZDE NELER ÖĞRENECEĞIZ?

- Programlama Nedir?
- Small Basic Nedir?
- Small Basic arayüzünü ve araçlarını tanıma
- «Merhaba Dünya» ile ilk Small Basic programımızı oluşturma
- Programı kaydetme ve tekrar açma

PROGRAMLAMA NEDIR?

- Programlama, makinaların bizleri anlamasını sağlamak için hazırlanan komut dizisidir.

PROGRAMLAMA NEDIR?

İnsanlar Türkçe, İngilizce, Fransızca ve İspanyolca gibi dilleri konuşarak anlaşır.

Bilgisayarlar da belirli dillerde yazılmış programları anlayabilirler.

SMALL BASIC NEDİR?

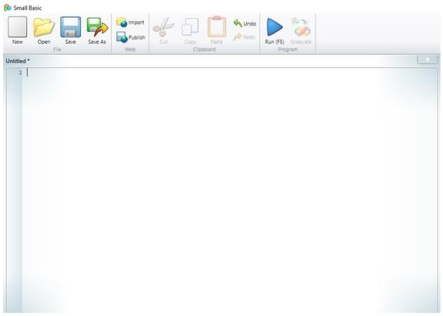
- Small Basic'te bilgisayarları konuşturabileceğimiz programlama dillerinden biridir.

SMALL BASIC NEDİR?

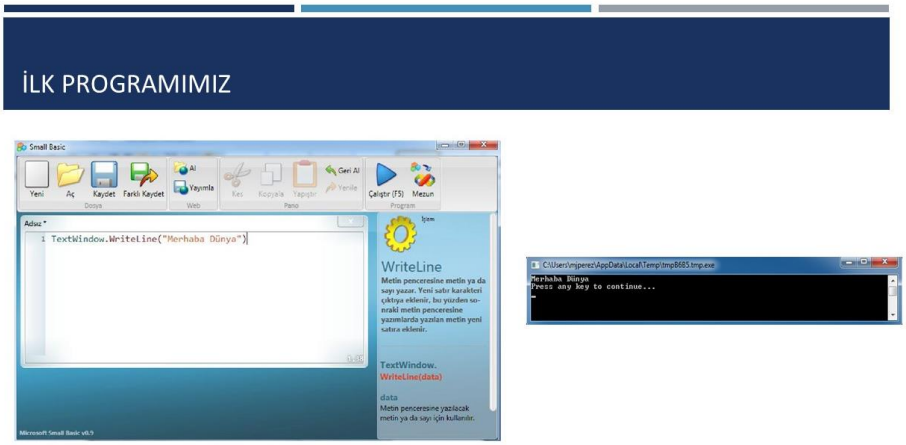
Microsoft® Small Basic

- Metinsel programlama dili olan Small Basic, Visual Basic ve Java gibi programlama dillerini öğrenmenizi kolaylaştıracaktır.

SMALL BASIC PROGRAMININ ARA YÜZÜ

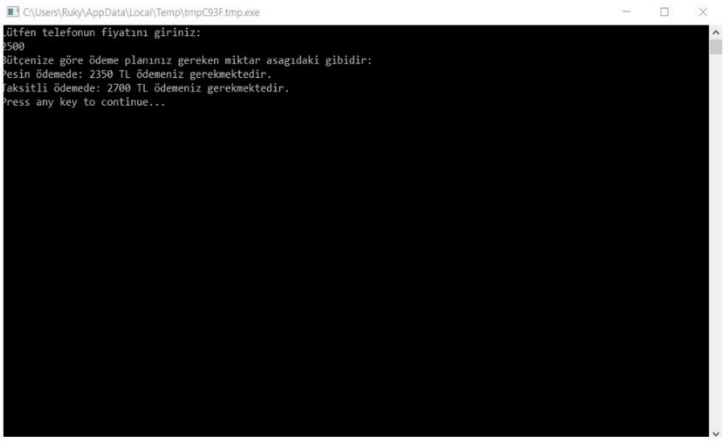


Hep beraber Small Basic ara yüzünü ve araçlarını tanıyalım. 😊



Week 3 For Experimental Group (Mathematics Integration)

Ali'nin alacağı 2500 TL değerindeki cep telefonu için mağaza 2 farklı ödeme seçeneği sunmuştur.
Ali telefonu 12 taksitle alırsa aylık ödemesi 225 TL olacakken, peşin alırsa satış fiyatı üzerinden 150 TL indirim kazanacaktır.
Telefonun taksitli ve peşin satış fiyatlarını hesaplayınız.



Week 3 For Control Group (Traditional)

```
C:\Users\raltin\AppData\Local\Temp\tmp3721.tmp.exe
Kahvaltıda ne yemek istersiniz?:
Tost ve portakal suyu
Merhaba! Tost ve portakal suyu sizin için en kısa zamanda hazır olacak.
Afiyet olsun.
Press any key to continue...
```

Week 4 For Experimental Group (Mathematics Integration)

Ali'nin alacağı 2500 TL değerindeki cep telefonu için mağaza 2 farklı ödeme seçeneği sunmuştur.
Ali telefonu 12 taksitle alırsa aylık ödemesi 225 TL olacakken, peşin alırsa satış fiyatı üzerinden 150 TL indirim kazanacaktır.
Telefonun taksitli ve peşin satış fiyatlarını hesaplayınız.

```
C:\Users\Ruky\AppData\Local\Temp\tmpBEDA.tmp.exe
Lütfen bütçenizi giriniz:
2700
Bütçeniz pesin alim için yeterli. Ödemeniz gereken miktar:2350 TL'dir
Press any key to continue...
```

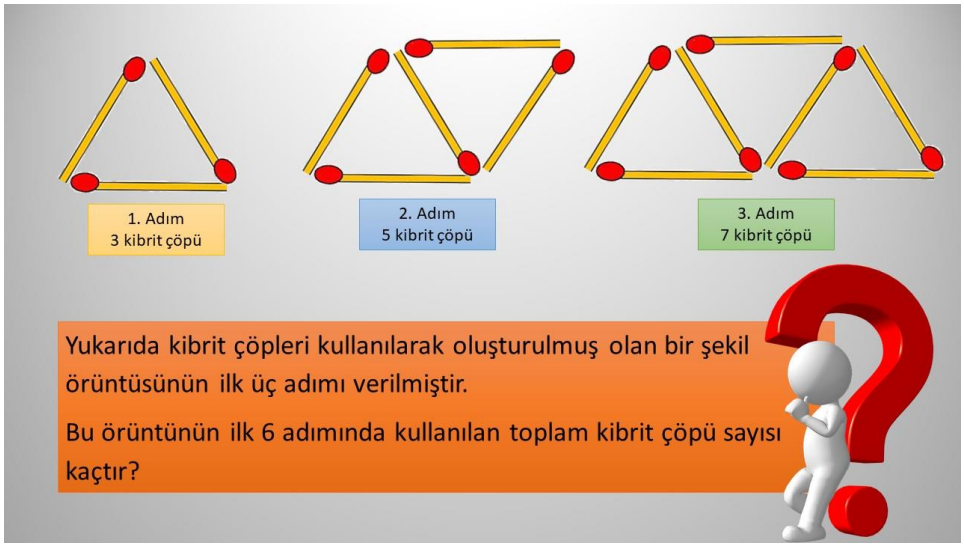
```
C:\Users\Ruky\AppData\Local\Temp\tmpAF55.tmp.exe
Lütfen bütçenizi giriniz:
2200
Bütçeniz pesin alim için yetersiz.
Taksitli almak isterseniz 12 ay süresi sonunda toplam ödemeniz gereken ücret: 2700 TL'dir.
Press any key to continue...
```

Week 4 For Control Group (Traditional)


```
C:\Users\raltin\AppData\Local\Temp\tmpDB4D.tmp.exe
Lütfen kahvaltıda ne yemek istediğinizi yazınız:

Kahvaltı tercihi yapmadınız.
 Dilerseniz aşağıdaki kahvaltı seçeneklerinden birini isteyebilirsiniz.
 1. Serpme Kahvaltı.
 2. Omlet.
 3. Tost.
 4. Simit .
Press any key to continue...
```

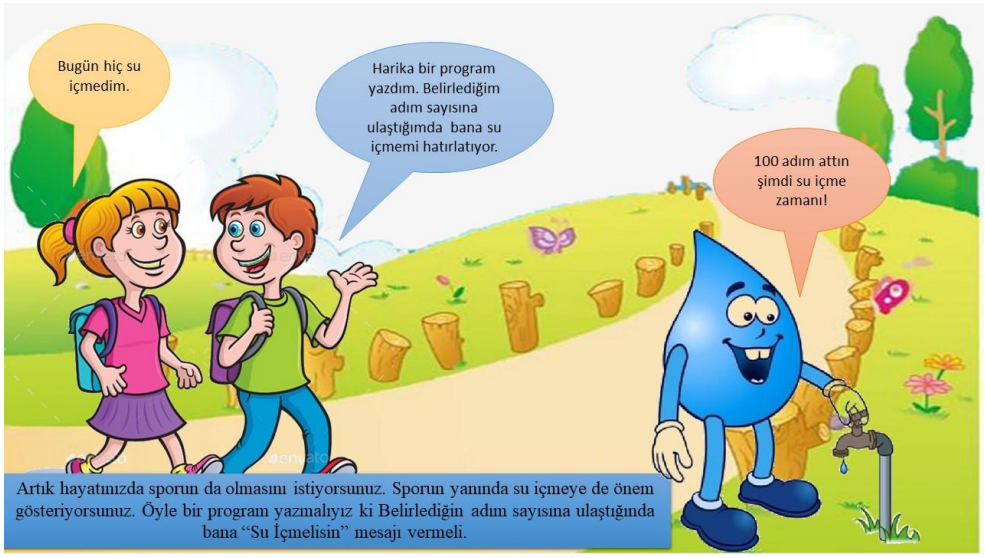
```
C:\Users\raltin\AppData\Local\Temp\tmp410D.tmp.exe
Lütfen kahvaltıda ne yemek istediğinizi yazınız:
Menemen
Merhaba! Menemen sizin için en kısa zamanda hazır olacak.
Afiyet olsun.
Press any key to continue...
```

Week 5 For Experimental Group (Mathematics Integration)



```
C:\Users\Ruky\AppData\Local\Temp\tmp60CC.tmp.exe
Lütfen örüntü adım sayısını giriniz:
6
n=1 için: 2 x 1 + 1 = 3
n=2 için: 2 x 2 + 1 = 5
n=3 için: 2 x 3 + 1 = 7
n=4 için: 2 x 4 + 1 = 9
n=5 için: 2 x 5 + 1 = 11
n=6 için: 2 x 6 + 1 = 13
Kullanılan toplam kibrit çöpü sayısı = 48
Press any key to continue...
```

Week 5 For Control Group (Traditional)



```
C:\Users\raltin\AppData\Local\Temp\tmp120C.tmp.exe
Kaç adım sonunda su içmelisin mesajı almak istersin?
10
Harikasın! 1 numaralı adımını attın
Harikasın! 2 numaralı adımını attın
Harikasın! 3 numaralı adımını attın
Harikasın! 4 numaralı adımını attın
Harikasın! 5 numaralı adımını attın
Harikasın! 6 numaralı adımını attın
Harikasın! 7 numaralı adımını attın
Harikasın! 8 numaralı adımını attın
Harikasın! 9 numaralı adımını attın
Harikasın! 10 numaralı adımını attın
100 adımı tamamladın. Şimdi bir bardak su içmelisin.
Press any key to continue...
```

Week 6 For Experimental Group (Mathematics Integration)

```
C:\Users\Ruky\AppData\Local\Temp\tmp85E2.tmp.exe
Lütfen takım adını giriniz: :
Lakers
 Takım skorunu giriniz :36
 Takım skorunu giriniz :42
 Takım skorunu giriniz :35
 Takım skorunu giriniz :40
 Takım skorunu giriniz :38
 Takım skorunu giriniz :
Lakers takımının oynadıgı maçlarda aldıgı skorların aritmetik ortalaması 38.2 seklindedir.
Press any key to continue...
```

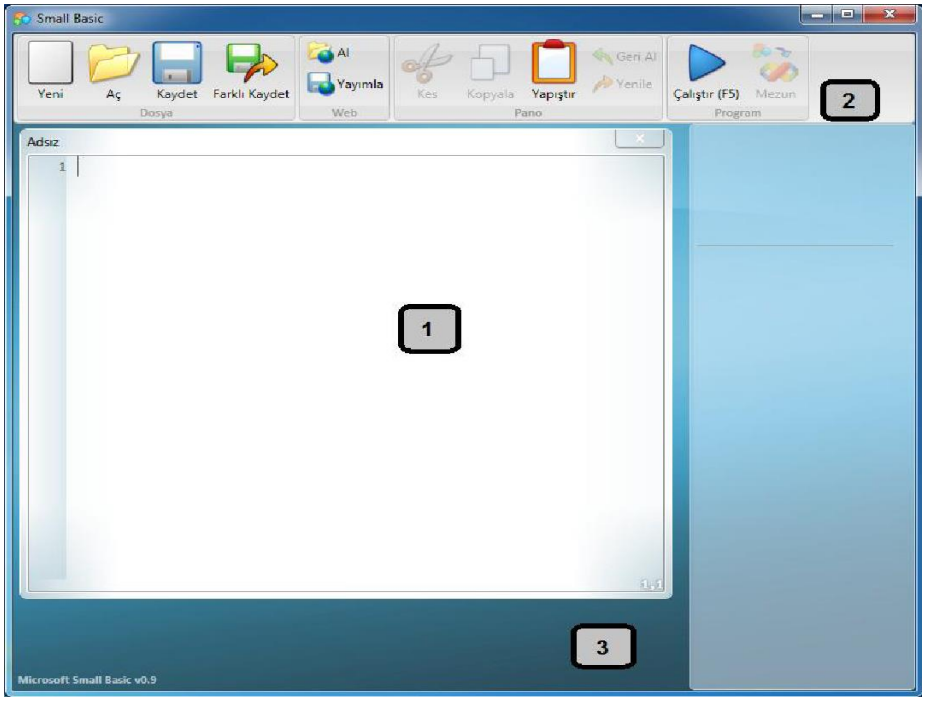
Week 6 For Control Group (Traditional)

```
C:\Users\raltin\AppData\Local\Temp\tmp3D15.tmp.exe
Lütfen isminizi giriniz:
Rukiye
Lütfen 1 ile 50 arasında bir tahmin giriniz:15
Yazdığın sayı küçük, biraz daha büyük tahiminlerde bulunmalısın.
Lütfen 1 ile 50 arasında bir tahmin giriniz:32
Yazdığın sayı büyük, biraz daha küçük tahiminlerde bulunmalısın.
Lütfen 1 ile 50 arasında bir tahmin giriniz:25
Harikasın Rukiye! Tuttuğum sayıyı tahmin edebildin!
Press any key to continue...
```

D. Guidance for Teachers

Small Basic User Interface

SmallBasic'i ilk çalıştırdığınızda, aşağıdaki şekle benzeyen bir pencere göreceksiniz.



Bu, Small Basic programlarımızı yazıp, çalıştıracağımız Small Basic Ortamıdır. Bu ortamın, sayılarla tanımlanmış çeşitli belirgin öğeleri vardır.

1. ile tanımlanmış olan **Düzenleyici**, Small Basic programlarını yazacağımız yerdir. Örnek bir programı ya da daha önce kaydedilmiş olan bir programı açtığınızda, ekranda bu düzenleyici görüntülenecektir. Bu durumda, o programı değiştirebilir ve daha sonra kullanmak üzere kaydedebilirsiniz.

   Ayrıca, bir kerede birden fazla programı açabilir ve üzerinde çalışabilirsiniz. Üzerinde çalıştığınız her bir program, ayrı bir düzenleyicide görüntülenecektir. O anda üzerinde çalıştığınız programı içeren düzenleyici, *aktif düzenleyici* olarak adlandırılır.

2. ile tanımlanmış olan **Araç Çubuğu**, *aktif düzenleyiciye* veya ortama komut vermek için kullanılır. İlerledikçe, araç çubuğundaki çeşitli komutları öğreneceğiz.

3. ile tanımlanmış olan **Yüzey**, tüm düzenleyici pencerelerinin gittiği yerdir.

E. Instruments

Mathematics Attitude Scale

MATEMATİK DERSİNE YÖNELİK TUTUM ÖLÇEĞİ

Bu ölçek sizin matematik dersiyle ilgili düşüncelerinizi öğrenmek için hazırlanmıştır. Cümlelerden hiçbirinin kesin cevabı yoktur. Her cümleyle ilgili görüş, kişiden kişiye değişebilir. Bunun için vereceğiniz cevaplar kendi görüşünüzü yansıtmalıdır. Her cümleyle ilgili görüş belirtirken önce cümleyi dikkatle okuyunuz, sonra cümlede belirtilen düşüncenin, sizin düşünce ve duygunuza ne derecede uygun olduğuna karar veriniz (Duatepe ve Çilesiz, 1999).

	Kesinlikle Katılmıyorum	Katılmıyorum	Kararsızım	Kısmen Katılıyorum	Tamamen Katılıyorum
1. Matematik beni korkutmuyor.					
2. Matematik sevdiğim dersler arasındadır.					
3. Matematik çalışmayı isterim.					

4. Matematiği hayatım boyunca bir çok yerde kullanacağım.					
5. Matematik çalışırken gergin olurum.					
6. Yeni bir matematik problemiyle uğraşırken kendimi rahat hissederim.					
7. Matematiği anlamaya çalışmak zaman kaybıdır.					
8. Matematik çalışmanın teşvik edici hiç bir yanı yok.					
9. Matematik öğrenmek zahmete değer.					
10. Matematik problemlerini çözmeye çalışmak bana çekici gelmiyor.					
11. Matematik çalışırken sıradışı bir soruyla karşılaşınca yanıt bulana kadar uğraşırım.					
12. Bu derste öğrendiklerimi günlük hayatta kullanacağımı sanmıyorum.					
	Kesinlikle Katılmıyorum	Katılmıyorum	Kararsızım	Kısmen Katılıyorum	Tamamen Katılıyorum
13. Bazı insanların matematikten nasıl bu kadar hoşlandıklarını anlamıyorum.					
14. Meslek hayatımda matematiği kullanacağımı düşünmüyorum.					
15. Zorunlu olmasam matematik derslerine girmezdim.					
16. Matematik çalışmaya başlayınca bırakmak zor gelir.					
17. Matematiği iyi bilmek çalışma olanaklarımı artıracaktır.					
18. Matematik derslerinde iyi notlar alabilirim.					
19. Matematik çalışırken kaygılı olmam.					
20. Matematiksel düşünme yeteneğine sahip değilim.					
21. Karşılaştığım problemleri matematik kullanarak çözmek hoşuma gider.					
22. Matematiği anlayamayacağımı düşünüyorum.					
23. Matematik bir bilim değil yalnızca bir araçtır.					
24. Derste çözümü yarım kalan matematik sorularıyla uğraşmak bana zevk verir.					
25. Matematik derslerinde başarılı olmak benim için önemlidir.					
26. Matematik çalışmak gerektiğinde kendime güvenmem.					
27. Matematik alanında iddialıyım.					

28. Başkalarıyla matematik hakkında konuşmaktan hoşlanmam.					
29. Matematik dersinden zevk alıyorum.					
30. Matematiğin adını bile duymak beni huzursuz eder.					
31. Bundan başka matematik dersi almak istemiyorum.					
32. Diğer dersler bana matematikten daha önemli gelir.					
33. Matematik kafarnı karıştırır.					
34. Matematik sıkıcıdır.					
35. Matematik en korktuğum derslerden biridir.					
36. Matematik çalışırken kendimi pçok çaresiz hissediyorum.					
37. Keşke diğer derslerde matematik kullanmam gerekmeseydi.					

Computational Thinking Skills Self-Efficacy Scale

Ortaokul Öğrencileri İçin Bilgi İşlemsel Düşünme Becerisine Yönelik Öz Yeterlik Algısı (BİDBÖA) Ölçeği

Sıra No	Soru Metni	Evet	Kısmen	Hayır
Algoritma Tasarlama Yeterliği				
1	Algoritmaların hangi amaçla kullanıldığını anlıyorum.	1	2	3
2	Algoritmanın ne olduğunu biliyorum.	1	2	3
3	Basit algoritmalar oluşturabilirim.	1	2	3
4	Koşullu algoritmalar oluşturabilirim.	1	2	3
5	Döngü yapısında algoritmalar oluşturabilirim.	1	2	3
6	Algoritma oluştururken mantıksal sorgulama yapabilirim.	1	2	3
7	Bir algoritmanın çıktısının ne olacağını tahmin edebilirim.	1	2	3
8	Algoritmada bulunan hataları ayıklayabilirim.	1	2	3
9	Algoritmaların dijital araçlar için nasıl koda dönüştürüleceğini anlıyorum.	1	2	3
Problem Çözme Yeterliği				
10	Problemi çözüp sonucunu bulduktan sonra yaptığım işlemleri kontrol ederim.	1	2	3
11	Problemi çözüp sonucunu bulduktan sonra yaptığım işlemleri kontrol eder varsa hataları düzeltirim.	1	2	3
12	Bir problemi okuduğumda, çözüm için hangi bilgiye ihtiyacım olduğunu düşünürüm.	1	2	3
13	Problem çözüm sürecinde işlem önceliklerine dikkat ederim.	1	2	3

14	Bir problemi okuduğumda, çözüm için gerekli ve gereksiz olan bilgiyi ayırt edebilirim.	1	2	3
15	Farklı çözüm yollarını inceleyerek daha iyi bir çözüm bulmaya çalışırım.	1	2	3
16	Bir problemi okuduğumda, daha önce çözdüğüm problemleri düşünerek benzerlik ve farklılıklarına göre aralarında ilişki kurarım.	1	2	3
17	Problem çözerken, hangi işlemi neden yaptığımı sürekli sorgularım.	1	2	3
18	Bir problemi çözebilmem için yeterli veri sunulup sunulmadığına karar verebilirim.	1	2	3
19	Bir problem için ürettiğim çözümü farklı problemlere genelleyebilirim.	1	2	3
Veri İşleme Yeterliği				
20	Verinin ne olduğunu biliyorum.	1	2	3
21	Veri toplamanın önemini anlıyorum.	1	2	3
22	Verinin farklı türleri olduğunun (sayı ve metin) farkındayım.	1	2	3
23	Veri ve bilgi arasındaki farklı açıklayabilirim.	1	2	3
24	Problemlerin çözümünde farklı veri türleri kullanılabileceğini biliyorum.	1	2	3
25	Verilerin tablo yapısında daha anlamlı sunulabildiğini biliyorum.	1	2	3
26	Dijital verinin farklı biçimlere dönüşebileceğini biliyorum.	1	2	3
Sıra No	**Soru Metni**	**Evet**	**Kısmen**	**Hayır**
Temel Programlama Yeterliği				
27	Değişkenleri tanımlayıp kullanabilirim.	1	2	3
28	Koşullu yapıları ve döngüleri oluştururken aritmetik operatörleri kullanabilirim.	1	2	3
29	Bir döngüyü sonlandırmak için değişken ve ilişkisel operatörleri kullanabilirim.	1	2	3
30	Farklı kontrol durumları için değişik döngüler oluşturabilirim.	1	2	3
31	Belirli işlemler için hazır fonksiyonları kullanabilirim.	1	2	3
Özgüven Yeterliği				
32	Yönergelerin ve işlem adımlarının önemini biliyorum.	1	2	3
33	Çözümleri göstermek için şemalar kullanabilirim.	1	2	3
34	Aynı problem için farklı çözümler üretilebileceğinin farkındayım.	1	2	3
35	Problem çözme sürecinde hatalarımı nasıl düzelteceğimi biliyorum.	1	2	3
36	Dijital araçlar tarafından en iyi başarılan işlemlerin ne olduğunun farkındayım.	1	2	3

Ölçek formu; 5 faktöre ayrılmış toplam 36 maddeden oluşmaktadır. Ortaokul düzeyinde, öğrenciler tarafından daha rahat karar verilmesine destek olmak amacıyla, 3'lü Likert yapıda desenlenmiştir. Bu yapı içerisinde; "Evet", 1 puana, "Kısmen", 2 puana ve "Hayır", 3 puana karşılık gelmektedir. Ölçekten alınacak toplam puan 36 ile 108 arasında değişmektedir.

İfadeler sıralı olarak, küçükten büyüğe doğru yer aldığı için, toplam puan hesaplanırken, alınan puanının düşük olması , bireyin *"Bilgi İşlemsel Düşünme Becerisine Yönelik Öz Yeterlik Algısı"* düzeyinin yüksek olduğuna ilişkin ipuçları verecektir. Ölçek maddeleri içerisinde, ters puanlama gerektiren bir ifade bulunmamaktadır. O nedenle, her faktör kendi içerisinde, verilen cevaplardan elde edilen toplam puan üzerinden değerlendirilebilmektedir. Son olarak, ölçekten elde edilen puanın sadece beceri algısına yönelik olduğu ve Bilgi işlemsel düşünme becerilerinin ölçülebilmesi için , bir ölçek formundan elde edilebilecek verilerden çok daha kapsamlı verilere ulaşılması gerektiği unutulmalıdır (Gülbahar et al., 2019).

Computer Programming Self-Efficacy Scale

Programlamaya İlişkin Öz Yeterlilik Algısı Ölçeği

	Kendime hiç Güvenmiyorum	Genellikle Güvenmiyorum	Biraz Güvenmiyorum	%50/%50	Oldukça Güveniyorum	Genellikle Güveniyorum	Tamamen Güveniyorum
"Merhaba Dünya" mesajının görüntülenebileceği bir program yazabilirim.							
Üç sayının ortalamasını hesaplayan bir program yazabilirim.							
Verilen herhangi bir sayı dizisinin ortalamasını hesaplayan bir program yazabilirim.							
İstenilenler açıkça tanımlandığında bir problemin çözümüne yönelik oldukça karmaşık ve uzun bir program yazabilirim							
Yazacağım bir programı modüler bir biçimde organize edip tasarlayabilirim							
Yazdığım uzun ve karmaşık bir programdaki tüm hataları ayıklayabilir ve çalışabilir hale getirebilirim.							
Uzun, karmaşık ve birden fazla dosya gerektiren bir programı kavrayabilirim.							
Bir programın daha okunabilir ve açık olması için uzun ve karmaşık kısımları yeniden yazabilirim.							
Çevrede bir sürü dikkat dağıtıcı olsa bile programa odaklanma yollarını bulabilirim.							

Small Basic Programming Achievement Test

Kişisel Bilgiler:

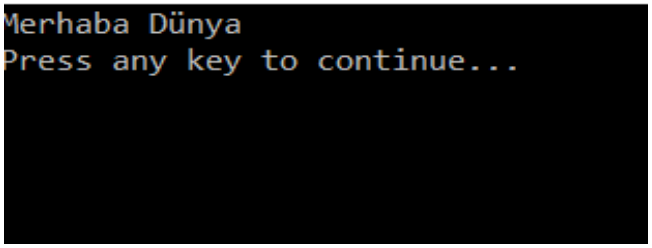
Öğrenci Numarası:

Cinsiyet:

- ☐ Erkek
- ☐ Kız

Bölüm 1 – Printing / Yazdırma

Soru 1:



Aşağıdaki kodlardan hangisi yukarıda verilen ekran görüntüsünü yansıtır?

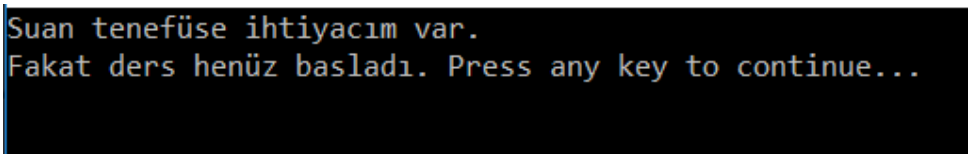
A. TextWindow.WriteLine ("Merhaba Dünya")
yazi= TextWindow.Read()

B. TextWindow.WriteLine ("Merhaba Dünya")

C. TextWindow.Write ("Merhaba Dünya")

D. TextWindow.ReadLine ("Merhaba Dünya")

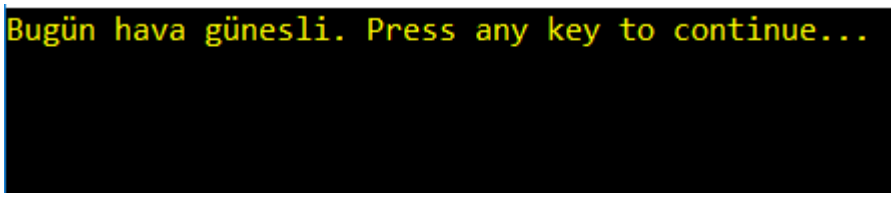
Soru 2:



Aşağıdaki kodlardan hangisi yukarıda verilen ekran görüntüsünü yansıtır?

A. TextWindow.Write ("Suan tenefüse ihtiyacim var.")
   TextWindow.WriteLine ("Fakat ders henüz basladi. ")
B. TextWindow.Write ("Suan tenefüse ihtiyacim var.")
   TextWindow.Write ("Fakat ders henüz basladi. ")
C. TextWindow.WriteLine ("Suan tenefüse ihtiyacim var.")
   TextWindow.Write ("Fakat ders henüz basladi. ")
D. TextWindow.WriteLine ("Suan tenefüse ihtiyacim var.")
   TextWindow.WriteLine ("Fakat ders henüz basladi. ")

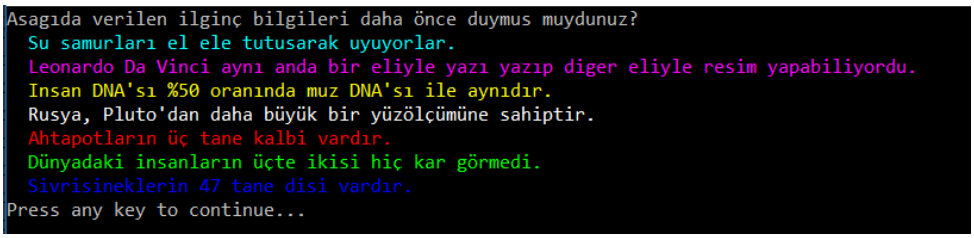
Soru 3:



Aşağıdaki kodlardan hangisi yukarıda verilen ekran görüntüsünü yansıtır?

A. TextWindow.BackgroundColor="yellow"
   TextWindow.WriteLine ("Bugün hava günesli. ")
B. TextWindow.Write ("Bugün hava günesli. ")
   TextWindow.BackgroundColor="yellow"
C. TextWindow.WriteLine ("Bugün hava günesli. ")
   TextWindow.ForegroundColor="yellow"
D. TextWindow.ForegroundColor="yellow"
   TextWindow.Write ("Bugün hava günesli. ")

Soru 4:



Yukarıdaki görüntü için aşağıda verilen kod yazılmıştır. Boşluk bırakılan yerlere aşağıdaki kodlardan hangisi gelmelidir?

```
TextWindow.[          ]("Asagida verilen ilginç bilgileri daha önce duymus muydunuz?")
TextWindow.[                ]="cyan"
TextWindow.[          ]("  Su samurlari el ele tutusarak uyuyorlar.")
TextWindow.[                ]="magenta"
TextWindow.[          ]("  Leonardo Da Vinci ayni anda bir eliyle yazi yazip diger eliyle resim yapabiliyordu.")
TextWindow.[                ]="yellow"
TextWindow.[          ]("  Insan DNA'si %50 oraninda muz DNA'si ile aynidir.")
TextWindow.[                ]="white"
TextWindow.[          ]("  Rusya, Pluto'dan daha büyük bir yüzölçümüne sahiptir.")
TextWindow.[                ]="red"
TextWindow.[          ]("  Ahtapotlarin üç tane kalbi vardir.")
TextWindow.[                ]="green"
TextWindow.[          ]("  Dünyadaki insanlarin üçte ikisi hiç kar görmedi.")
TextWindow.[                ]="blue"
TextWindow.[          ]("  Sivrisineklerin 47 tane disi vardir.")
TextWindow.[                ]="gray"
```

A. ForegroundColor
Write

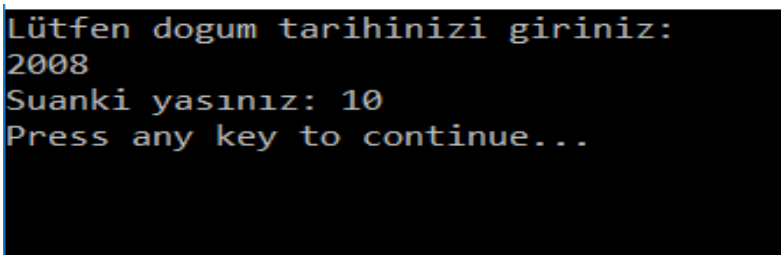
B. BackgroundColor
Write

C. BackgroundColor
WriteLine

D. ForegroundColor
WriteLine

Bölüm 2: Variables and User Input/ Değişkenler ve Kullanıcı Girişi

Soru 5:



Aşağıdaki kodlardan hangisi değişken kullanarak yukarıda verilen ekran görüntüsünü yansıtır?

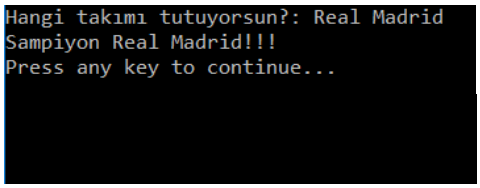
A.
```
TextWindow.WriteLine("Lütfen dogum tarihinizi giriniz: ")
        tarih = TextWindow.Read()
        yas = 2018 - tarih
TextWindow.WriteLine("Suanki yasiniz: " + yas)
```

B.
```
TextWindow.WriteLine("Lütfen dogum tarihinizi giriniz: ")
TextWindow.WriteLine("2018")
TextWindow.WriteLine("Suanki yasiniz: 10")
```

C.
```
TextWindow.Read("Lütfen dogum tarihinizi giriniz: ")
TextWindow.Read("2018")
TextWindow.Read("Suanki yasiniz: 10")
```

D.
```
TextWindow.WriteLine("Lütfen dogum tarihinizi giriniz: ")
        tarih = TextWindow.Read()
TextWindow.WriteLine("Suanki yasiniz: " + 2018-tarih)
```

Soru 6:




```
TextWindow.Write("Hangi takimi tutuyorsun?: ")
[ 1 ]
TextWindow.WriteLine("Sampiyon " [ 2 ]
```

Solda verilen ekran görüntüsünün kodu yukarıdaki gibidir. 1 ve 2 numaralı boşluklara aşağıdaki kodlardan hangisi gelmelidir?

A.
```
1) takim = TextWindow.Write()
2) +takim+ "!!!"
```
B.
```
1) takim = TextWindow.Read()
2) +takim+ "!!!"
```
C.
```
1) takim = TextWindow.Read()
2) +takim
```
D.
```
1) takim = TextWindow.Write()
2) +takim
```

Soru 7:

……. programlama dilinde bilgisayarın hafızasında tutması için kullanılacak verilere yapılan tanımlamadır.

Yukarıda verilen tanımlama aşağıdaki seçeneklerden hangisine aittir?

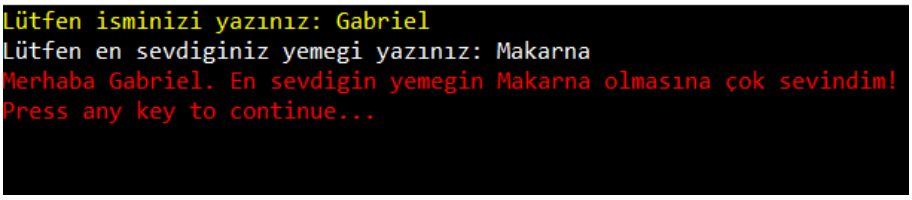
A. Algoritma (Algorithm)

B. Değişken (Variable)

C. Yazılım (Software)

D. Kod (Code)

Soru 8:



Yukarıdaki ekran görüntüsünü elde edebilmek için aşağıda verilen kodlar kullanılmıştır. Karışık olarak verilen kodlar ekrandaki görüntüyü elde edebilmek için seçeneklerden hangisinde doğru sıraya göre yazılmıştır.

I	TextWindow.ForegroundColor = "red"
II	TextWindow.Write("Lütfen en sevdiginiz yemegi yaziniz: ")
III	TextWindow.ForegroundColor = "yellow"
IV	TextWindow.Write("Lütfen isminizi yaziniz: ")
V	yemek = TextWindow.Read()
VI	isim = TextWindow.Read()
VII	TextWindow.WriteLine("Merhaba " +isim+ ". En sevdigin yemegin " +yemek+ " olmasina çok sevindim
VIII	TextWindow.ForegroundColor = "white"

A. IV – III – VIII – VI – V – II – VII - I

B. III – IV – VI – VIII – II – V – I - VII

C. III – IV – VIII – II – I – VII – VI - V

D. VI – V - III – IV – VIII – II – I – VII

Bölüm 3 – If Statements / Koşul

Soru 9:

Sıfırdan farklı olarak girilen bir sayının pozitif mi negative mi olduğunu bulmak için aşağıdaki komutlardan hangisini kullanmamız gerekmektedir?

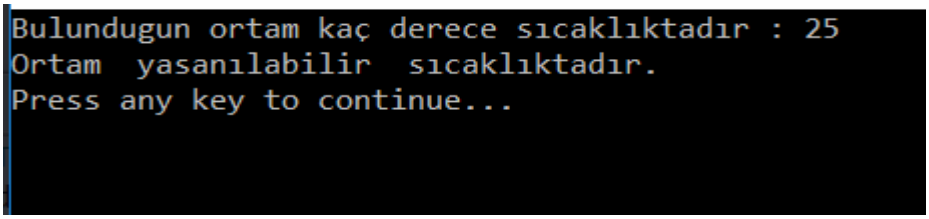
A. For Döngüsü

B. If-Else

C. While Döngüsü

D. Hiçbiri

Soru 10:



Yukarıda verilen ekran görüntüsünün kodlaması aşağıdaki gibi yapılmıştır. Programın doğru çalışabilmesi için iki satırın yer değiştirmesi gerekmektedir. Bu satırlar aşağıdaki seçeneklerden hangisidir?

I TextWindow.Write("Bulunduğun ortam kaç derece sıcaklıktadır : ")

II derece = TextWindow.Read()

III Else

IV TextWindow.WriteLine ("Ortam yaçanilabilir sıcaklıktadır. ")

V If derece >=20 And derece <=26 then

VI TextWindow.WriteLine ("Isiyı 20 ve 26 derece arasinda tutmanız gerekmektedir.")

VII EndIf

A. III - V

B. IV - VI

C. I - II

D. III - VII

Soru 11:

Pazar günü arkadaşları ile vakit geçirmek isteyen Javi arkadaşı Merve'den hava durumunu control etmesini ister. Javi Merve'ye üç seçenek sunmaktadır. Eğer hava

- Güneşli ise pikniğe,
- Yağmurlu ise Bowling'e
- Kapalı fakat yağış yok ise spora

Gidebileceklerini söyler.

Yukarıda verilen olayın kodlanmasında kaç tane if koşulu kullanılmalı?

Not: Yanlış girilen rakam için de hata mesajı vermeniz gerekmektedir.

A. 1

B. 2

C. 3

D. 4

Soru 12:

Girdiğimiz sınav notuna göre bize mesaj vermesini istediğimiz bir program yazmak istiyoruz. Notlara göre verilecek olan mesajların koşulları aşağıdaki gibidir;

- Eğer girilen not **85-100** arasında ise (85 ve 100 dahil) program "**Tebrikler! Bu sekilde ilerlemeye devam etmelisin.**" mesajını**,**
- Eğer girilen not **70-84** arasında ise (70 ve 84 dahil) program "**Biraz daha gayretle iyi sonuçlar alacagindan eminim.**" mesajını**,**
- Eğer girilen not **55-69** arasında ise (55 ve 69 dahil) program **"Daha fazla çalisman gerekmektedir."** mesajını**,**

- Eğer girilen not **54'ten küçük** (54 dahil) program **"Disiplinli çalisirsan çok daha iyi sonuçlar alacagina inaniyorum."** Mesajını vermeil.

Yukarıda verilen sorunun programlaması aşağıdaki gibidir.

```
TextWindow.Write("Lütfen matematik 1. sinav notunuzu giriniz: ")
not = TextWindow.Read()
If not >=85 And not<=100 then
  TextWindow.WriteLine ("Tebrikler! Bu sekilde ilerlemeye devam etmelisin.")
ElseIf not >=70 And not<=84 then
  TextWindow.WriteLine ("Biraz daha gayretle iyi sonuçlar alacagindan eminim.")
  ElseIf not >=55 And not<=69 then
    TextWindow.WriteLine ("Daha fazla çalisman gerekmektedir. ")
  Else
TextWindow.WriteLine ("Disiplinli çalisirsan çok daha iyi sonuçlar alacagina inaniyorum.")
EndIf
```

Programımızın 0'dan küçük ve 100'den büyük girilen not için "Lütfen geçerli bir not giriniz." Mesajını vermesini istiyoruz. Bunun için kırmızı kutu içine alınmış olan bölüm aşağıdkai seçeneklerden hangisi ile değiştirilirse programımız istediğimiz mesajı verir?

A.
```
If not>=0 and not<=54
   TextWindow.WriteLine ("Disiplinli çalisirsan çok daha iyi sonuçlar alacagina inaniyorum.")
 Else
   TextWindow.WriteLine ("Lütfen geçerli bir not giriniz.")
EndIf
```

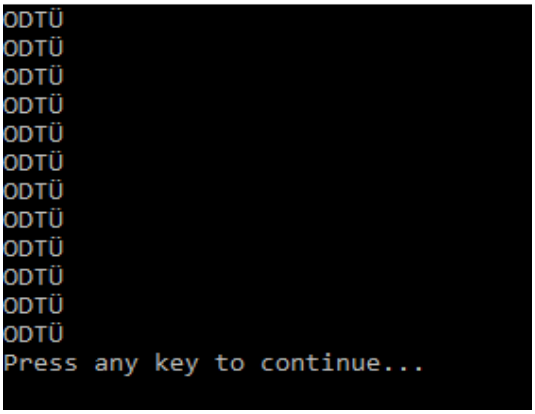
B.
```
If not<=54
   TextWindow.WriteLine ("Disiplinli çalisirsan çok daha iyi sonuçlar alacagina inaniyorum.")
 Else
   TextWindow.WriteLine ("Lütfen geçerli bir not giriniz.")
EndIf
```

C.
```
 Else

TextWindow.WriteLine ("Disiplinli çalisirsan çok daha iyi sonuçlar alacagina inaniyorum.")
TextWindow.WriteLine ("Lütfen geçerli bir not giriniz.")
EndIf
```

D.
```
ElseIf not>=0 and not<=54
   TextWindow.WriteLine ("Disiplinli çalisirsan çok daha iyi sonuçlar alacagina inaniyorum.")
 Else
   TextWindow.WriteLine ("Lütfen geçerli bir not giriniz.")
EndIf
```

Bölüm 4 – For Loop / For Döngüsü

Soru 13:



Yanda verilen ekran görüntüsünü almak için 12 kere TextWindow.WriteLine("ODTÜ") kodunun yazılması gerekmektedir. Aşağıda verilen program kodlarından hangisi ile aynı sonucu alabiliriz?

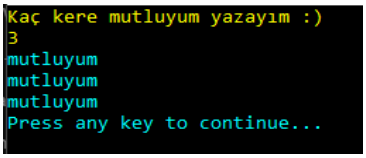
A.
```
For i = 1 To 12
  TextWindow.WriteLine("ODTÜ")
EndFor
```

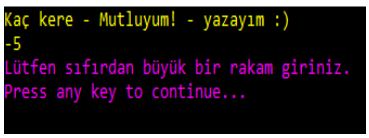
B.
```
yazi = "ODTÜ"
If i=12 Then
  TextWindow.WriteLine(yazi)
EndIf
```

C.
```
i = "ODTÜ"
For i = 1 To 12
  TextWindow.WriteLine(i)
EndFor
```

D.
```
If i=12 Then
  TextWindow.WriteLine("ODTÜ")
EndIf
```

Soru 14:





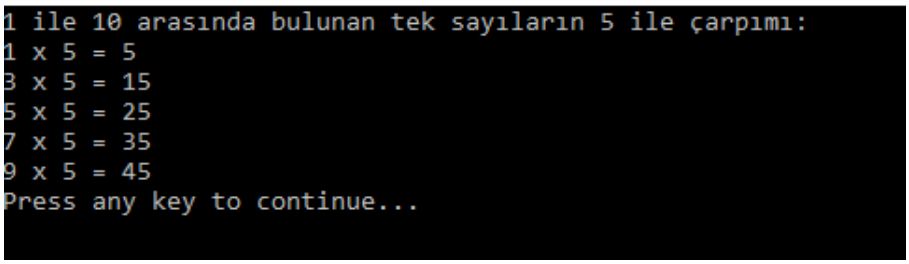

```
TextWindow.ForegroundColor = "yellow"
TextWindow.WriteLine ("Kaç kere - Mutluyum! - yazayım :)")
        1
        2
        3
TextWindow.ForegroundColor = "cyan"
  TextWindow.WriteLine ("Mutluyum!")
 EndFor
Else
 TextWindow.ForegroundColor = "magenta"
 TextWindow.WriteLine ("Lütfen sıfırdan büyük bir rakam giriniz.")
EndIf
```

Yukarıda verilen örnekte kullanıcının girdiği sayı kadar ekrara "**Mutluyum!**" mesajı gelmektedir. Eğer kullanıcı sıfırdan küçük bir rakam girerse, program "**Lütfen sıfırdan büyük bir rakam giriniz.**" mesajını vermektedir.

Programımızın doğru çalışması için yukarıda eksik verilen 1 - 2 ve 3 numaralı kutulara aşağıdaki komutlardan hangisi yazılmalı

A.
```
1- istek= TextWindow.Readnumber()
2- For i=1 To istek
3-  If ( istek >=1) Then
```
B.
```
1- For i=1 To istek
2- istek= TextWindow.Readnumber()
3- If ( istek >=1) Then
```
C.
```
1- If ( istek >=1) Then
2- For i=1 To istek
3- istek= TextWindow.Readnumber()
```
D.
```
1- istek= TextWindow.Read()
2- If ( istek >=1) Then
3-  For i=1 To istek
```

Soru 15:



Aşağıda verilen kodlardan hangisi ile yukarıda verilen ekran görüntüsünü elde edebiliriz?

Not: Step komutu rakamlar arası istediğiniz aralıklarla artırma/azaltma yapabilmenizi sağlar. Örneğin rakamlarda **5'er 5'er artış** olmasını istiyorsanız **Step 5** komutunu kullanmanız gerekmektedir.

A.
```
TextWindow.WriteLine("1 ile 10 arasında bulunan tek sayıların 5 ile çarpımı:")
For i = 1 to 10 Step 2
TextWindow.WriteLine(i + " x " + sayi + " = " + i * sayi)
EndFor
```

B.
```
TextWindow.WriteLine("1 ile 10 arasında bulunan tek sayıların 5 ile çarpımı:")
sayi = 5
For i = 1 to 10 Step 2
TextWindow.WriteLine(i + " x " + sayi + " = " + i * sayi)
EndFor
```

C.
```
TextWindow.WriteLine("1 ile 10 arasında bulunan tek sayıların 5 ile çarpımı:")
sayi = 5
For i = 1 to 10 Step 5
TextWindow.WriteLine(i + " x " + sayi + " = " + i * sayi)
EndFor
```

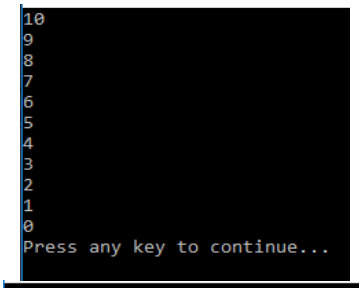
D.
```
TextWindow.WriteLine("1 ile 10 arasında bulunan tek sayıların 5 ile çarpımı:")
sayi = 5
For i = 1 to 10
TextWindow.WriteLine(i + " x " + sayi + " = " + i * sayi)
    EndFor
```

Soru 16:

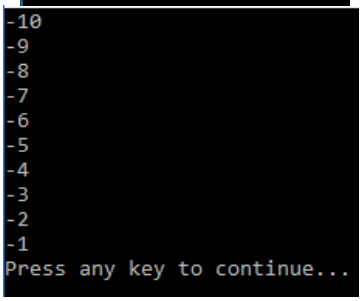
```
For i = 10 to 0 Step -1
TextWindow.WriteLine(i)
EndFor
```

Aşağıda verilen kod aşağıdaki ekran görüntülerinden hangisini verir?

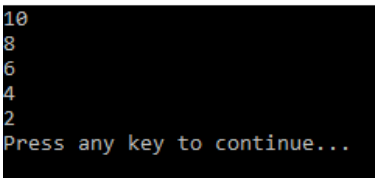
A.



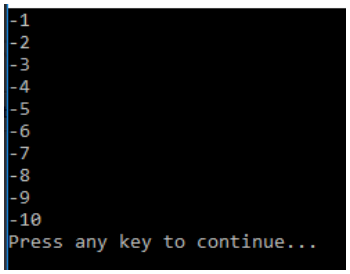
B.



C.



D.



Bölüm 4 – While Loop / While Döngüsü

Soru 17:

```
sayi = 100
While (sayi>1)
  TextWindow.WriteLine(sayi)
  sayi=sayi/2
Endwhile
```

Yukarıda verilen programın yorumlaması aşağıdakilerden hangisidir?

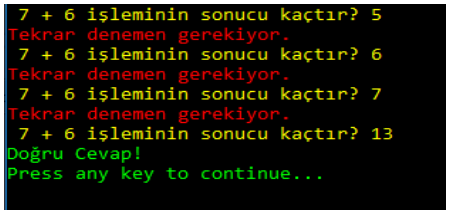
A. Program 100'ü bulana kadar döngüden gelen rakamı ikiye bölüp yazdırır.

B. Program döngüden gelen her rakamı ikiye böler ve sonsuza kadar devam eder.

C. Program Sayı 1'e ulaşıncaya kadar 100'den başlayarak döngüden gelen rakamı ikiye bölüp yazdırırır.

D. Program Sayı 1'den büyük olduğu sürece 100'den başlayarak döngüden gelen rakamı ikiye bölüp yazdırırır.

Soru 18:



Solda verilen ekran görüntüsünde program belli bir koşula ulaşana kadar tekrar ediyor. Bu koşula göre programın kodları aşağıdakilerden hangisidir?

A.
```
For cevap > 13 to 0
 TextWindow.ForegroundColor = "yellow"
 TextWindow.Write(" 7 + 6 işleminin sonucu kaçtır? ")
 cevap = TextWindow.ReadNumber()
 If (cevap = 13) Then
   TextWindow.ForegroundColor = "green"
   TextWindow.WriteLine("Doğru Cevap!")
 Else
  TextWindow.ForegroundColor = "red"
  TextWindow.WriteLine("Tekrar denemen gerekiyor.")
 EndIf
EndFor
```

B.
```
For cevap > 13 to 0
 TextWindow.ForegroundColor = "yellow"
 TextWindow.Write(" 7 + 6 iþleminin sonucu kaçtýr? ")
 cevap = TextWindow.ReadNumber()
   TextWindow.ForegroundColor = "green"
   TextWindow.WriteLine("Doðru Cevap!")
  TextWindow.ForegroundColor = "red"
  TextWindow.WriteLine("Tekrar denemen gerekiyor.")
EndFor
```

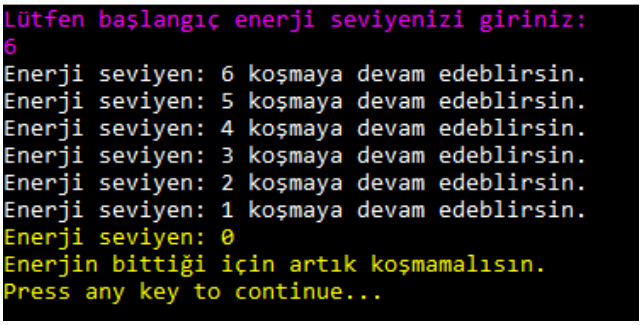
C.
```
While (cevap <> 13)
```

```
  TextWindow.ForegroundColor = "yellow"
  TextWindow.Write(" 7 + 6 iþleminin sonucu kaçtýr? ")
  cevap = TextWindow.ReadNumber()
  If (cevap = 13) Then
    TextWindow.ForegroundColor = "green"
    TextWindow.WriteLine("Doðru Cevap!")
  Else
    TextWindow.ForegroundColor = "red"
   TextWindow.WriteLine("Tekrar denemen gerekiyor.")
  EndIf
EndWhile
```

D.

```
While (cevap <> 13)
  TextWindow.ForegroundColor = "yellow"
  TextWindow.Write(" 7 + 6 iþleminin sonucu kaçtýr? ")
  cevap = TextWindow.ReadNumber()
    TextWindow.ForegroundColor = "green"
    TextWindow.WriteLine("Doðru Cevap!")
   TextWindow.ForegroundColor = "red"
   TextWindow.WriteLine("Tekrar denemen gerekiyor.")
EndWhile
```

Soru 19:



Yukarıda verilen ekran görüntüsünün programlaması aşağıda karışık bir şekilde verilmiştir. Hangi sıra ile yazılırsa programın çıktısı yukarıdaki gibi olur?

I
```
TextWindow.ForegroundColor ="white"
TextWindow.WriteLine("Enerji seviyen: " +enerji + " koşmaya devam edeblirsin.")
```

II
```
enerji=enerji-1
```

III
```
If enerji>0 Then
```

IV
```
enerji = TextWindow.Read()
enerji = enerji+1
```

V
```
While enerji >0
```

VI
```
TextWindow.ForegroundColor ="magenta"
TextWindow.WriteLine("Lütfen başlangıç enerji seviyenizi giriniz: ")
```

VII
```
EndIf
```

VIII
```
EndWhile
```

IX
```
Else
```

X
```
TextWindow.ForegroundColor ="yellow"
TextWindow.WriteLine ("Enerji seviyen: " +enerji)
TextWindow.WriteLine("Enerjin bittiği için artık koşmamalısın.")
```

A. VI-IV-V-II-III-I-IX-X-VII-VIII

B. I-IV-III-I-X-XI-VII-VIII-VI-II

C. XI-VII-III-I-VI-II-VIII-I-X-IV

D. VI-IV-V-II-III-I-X-XI-VII-VIII

Soru 20:

```
TextWindow.WriteLine("Lütfen bir rakam giriniz: ")
rakam = TextWindow.Read()
rakam = rakam * 10
          ?
  TextWindow.WriteLine(rakam)
  rakam = rakam / 5
EndWhile
```

Yukarıda verilen programda girilen rakamın önce 10 ile çarpımı hesaplanmaktadır. Sonrasında ise döngüden alınan sayının ekrana rakam 1'den küçük oluncaya kadar 5'e bölünerek yazdırılması istenmektedir. Programın doğru çalışması için soru işaretli bölüme aşağıdaki seçeneklerden hangisi gelmeli?

A. `While (rakam < 1)`

B. `While (rakam < 0)`

C. `While (rakam > 1)`

D. `While (rakam > 0)`

Interview Protocol for Students

1. Cinsiyetiniz

a) Kız
b) Erkek

2. Yaşınız: .....

3. Okulunuz: .....

4. Sınıfınız: .....

5. Daha önce herhangi bir programlama dili kullandın mı? Evet ise hangileri olduğunu söyleyebilir misin?

6. Small Basic programı ile ilgili görüşlerini alabilir miyim?

a) Negatif yönleri:
b) Pozitif yönleri:

7. Small Basic programını öğrenirken seni en çok hangi bölüm/bölümler zorladı?

8. Small Basic programını öğrenirken, öğrenmende yardımcı olan unsur/unsurlar nelerdi?

9. Small Basic programı öğrenimi ile ilgili belirtmek istediğin bir şey var mı?

Sorularım bitti. Sizin sormak istediğiniz bir şey yoksa görüşmeyi bitirebiliriz. Görüşmeye katıldığınız ve görüşlerinizi benimle paylaştığınız için teşekkür ederim.

Interview Protocol for Teachers

İlk başta sizi tanımak istiyorum.

c) Ne kadar süredir bu okulda çalışıyorsunuz?
d) Ne kadar süredir bilgisayar derslerine giriyorsunuz?
e) Daha önce herhangi bir programlama dilini çocuklara öğrettiniz mi?

(Evet ise)

i. Hangi programlama dillerini öğretiyorsunuz?
ii. Programlama dillini kullanma amaçlarınız neler?
iii. Programlama dillerini öğretirken zorluk yaşıyormusunuz? Evet ise bu zorluklar neler?

2. Matematik dersi ile işbirliği yapmak öğrencilerin programlamaya yönelik motivasyonuna etkiledi mi?

a) Etkilediği düşünüyorsanız nedenlerini açıklayabilir misiniz?
b) Etkilemediğini düşünüyorsanız nedenlerini açıklayabilir misiniz?

3. Matematik işbirliği ve klasik yöntem ile anlatılan dersleri kıyaslayarak aşağıdaki sorularını yanıtlayabilir misiniz?

c) Matematik ile anlatılan dersin artıları nelerdi?
d) Matematik ile anlatılan dersin eksileri nelerdi?
e) Klasik yöntem ile anlatılan dersin artıları nelerdi?
f) Klasik yöntem ile anlatılan dersin eksileri nelerdi?
g) Gelecekte hangi yöntem ile anlatmayı tercih edersiniz? Neden?

4. Bu dersi yaklaşık **6** hafta **12** saat olarak uygulanan bu derste gördüğünüz eksiklikler nelerdi?

5. Tasarlanan ders planlarında olması gerektiğini düşündüğünüz önerilerinizi paylaşabilir misiniz?

6. Matematiğin disiplinler arası iş birliği ile kullanılarak programlama öğretiminde yer alması hakkındaki görüş ve önerileriniz nelerdir?

Sorularım bitti. Sizin sormak istediğiniz bir şey yoksa görüşmeyi bitirebiliriz. Görüşmeye katıldığınız ve görüşlerinizi benimle paylaştığınız için teşekkür ederim.

F. Teachers' Effect

Results for MAS:

Within Group Comparisons

Table F.1 Paired Samples t-Tests (by Teachers)

Pair	*N*	*M*	*SD*	*t*	*df*	*p*
Teacher 1: Experimental group (Pretest)	46	3.07	0.32	**4.238**	**45**	**.00****
Teacher 1: Experimental group (Posttest)	46	2.83	0.18			
Teacher 2: Experimental group (Pretest)	48	2.99	0.22	**2.472**	**47**	**.02***
Teacher 2: Experimental group (Posttest)	48	2.9	0.15			
Teacher 1: Control group (Pretest)	47	2.88	0.19	-0.99	46	.33
Teacher 1: Control group (Posttest)	47	2.91	0.13			
Teacher 2: Control group (Pretest)	47	2.89	0.29	**2.313**	**46**	**.03***
Teacher 2: Control group (Posttest)	47	2.76	0.16			

Between Group Comparisons

Table F.2 Independent Sample t-Tests (by Teachers)

Measurement	Pair	*N*	*M*	*SD*	*t*	*df*	*p*
Mathematics Attitude Scale (Pretest)	Teacher 1	93	2.97	0.28	0.43	184	.67
	Teacher 2	93	2.96	0.23			
Mathematics Attitude Scale (Posttest)	Teacher 1	92	2.87	0.16	1.56	184	.12
	Teacher 2	94	2.84	0.17			

Covariance Analysis

Table F.3 Covariance Analysis of Mathematics Attitude Scale Pretest–Posttest Scores (Students of Teacher 1, Teacher 2)

Source	*SS*	*df*	*MS*	*F*	*p*	η_p^2
MAS Pre-Test	0.007	1	0.007	0.281	.596	.002
Teacher	0.081	1	0.081	3.156	.077	.017
Error	4.668	181	0.026			

[a.] R Squared = .019 (Adjusted R Squared = .008)

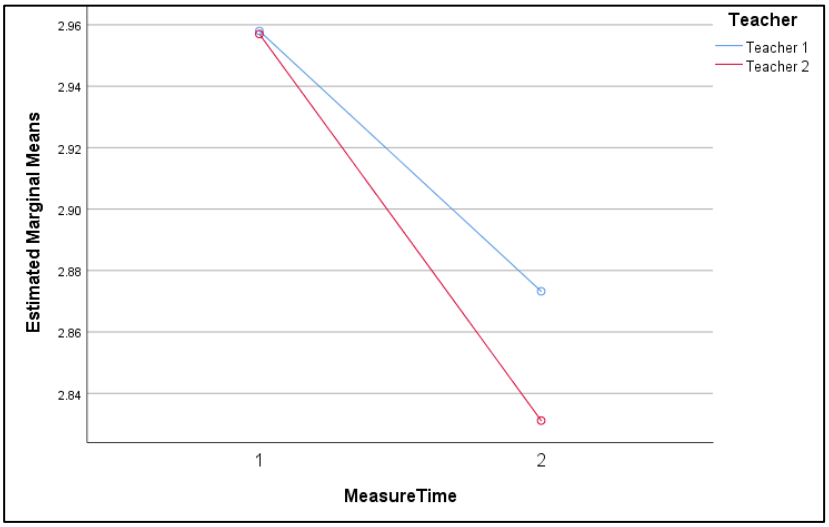


Figure F.1. Effect of Pretest Scores and Teachers' Attitude on Mathematics Attitude Scale Posttest Scores

Results for CTSSES:

Within Group Comparisons

Table F.4. Paired Samples t-Tests (by Teachers)

Pair	*N*	*M*	*SD*	*t*	*df*	*p*
Teacher 1: Experimental group (Pretest)	42	43.81	10.85	**-8.488**	**41**	**.00****
Teacher 1: Experimental group (Posttest)	42	61.79	6.03			
Teacher 2: Experimental group (Pretest)	48	37.14	15.80	**-9.447**	**47**	**.00****
Teacher 2: Experimental group (Posttest)	48	59.85	7.37			
Teacher 1: Control group (Pretest)	47	44.9	17.14	0.069	46	.95
Teacher 1: Control group (Posttest)	47	44.66	13.40			
Teacher 2: Control group (Pretest)	47	44.24	15.21	0.95	46	.35
Teacher 2: Control group (Posttest)	47	41.96	13.62			

Covariance Analysis

Table F.5. Covariance Analysis of Computational Thinking Skills Self-Efficacy Scale Pretest-Posttest Scores (Students of Teacher 1 & Teacher 2)

Source	*SS*	*df*	*MS*	*F*	*p*	η_p^2
CTSSES Pre-Test	173.929	1	173.929	.905	.343	.005
Teacher	177.777	1	177.777	.925	.337	.005
Error	34,769.127	181	192.095			

[a.] R Squared = .009 (Adjusted R Squared = -.002)

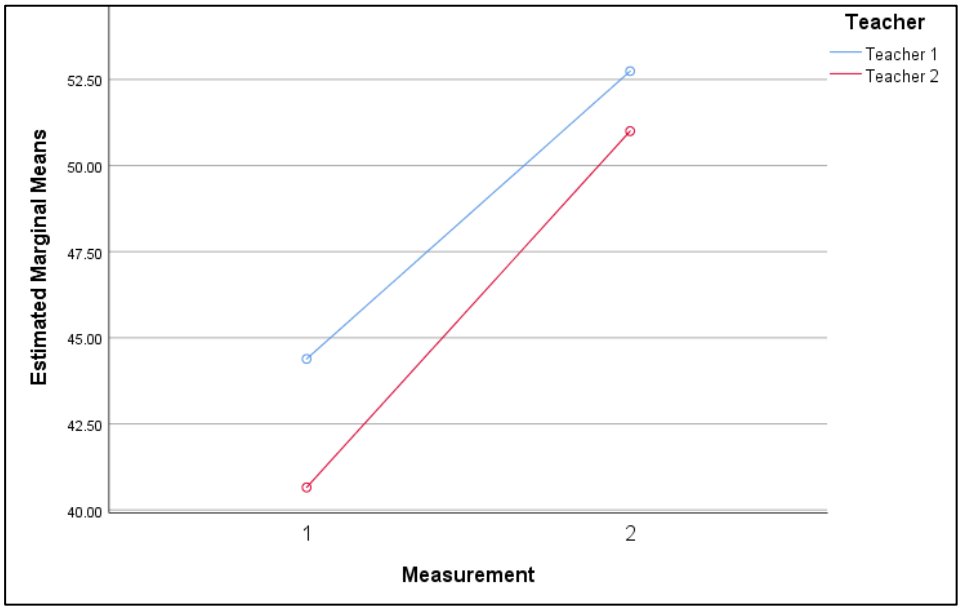


Figure F.2. Effect of Pretest Scores and Teachers' Attitude on Computational Thinking Skills Self-Efficacy Scale Posttest Scores

Results for CPSES:

Within Group Comparisons

Table F.6 Paired Sample t-Tests (by Teachers)

Pair	*N*	*M*	*SD*	*t*	*df*	*p*
Teacher 1: Experimental group (Pretest)	44	6.13	2.33	**-11.93**	**43**	**.00****
Teacher 1: Experimental group (Posttest)	44	11.00	1.40			
Teacher 2: Experimental group (Pretest)	46	5.91	2.38	**-13.86**	**45**	**.00****
Teacher 2: Experimental group (Posttest)	46	11.68	1.36			
Teacher 1: Control group (Pretest)	45	8.93	2.99	**2.49**	**44**	**.02***
Teacher 1: Control group (Posttest)	45	7.63	1.84			

Teacher 2: Control group (Pretest)	47	7.33	2.77	-1.75	46	.09
Teacher 2: Control group (Posttest)	47	8.12	2.34			

Between Group Comparisons

Table F.7 Independent Sample t-Tests (by Teachers)

Scale (Test)	Group	*N*	*M*	*SD*	*t*	*df*	*p*
CPSES-PreTest	Teacher 1	93	7.64	3.01	**2.55**	**186**	**.02***
	Teacher 2	95	6.58	2.67			
CPSES-PostTest	Teacher 1	89	9.30	2.35	-1.58	180	.11
	Teacher 2	93	9.88	2.61			

Covariance Analysis

Table F.8 Covariance Analysis of Computer Programming Self-Efficacy Scale Pretest–Posttest Scores (Students of Teacher 1 & Teacher 2)

Source	*SS*	*df*	*MS*	*F*	*p*	η_p^2
CPSES Pre-Test	44.785	1	44.785	**7.492**	**.007****	.040
Teacher	7.916	1	7.916	1.324	.251	.007
Error	1,070.051	179	5.978			

[a.] R Squared = .053 (Adjusted R Squared = .043)

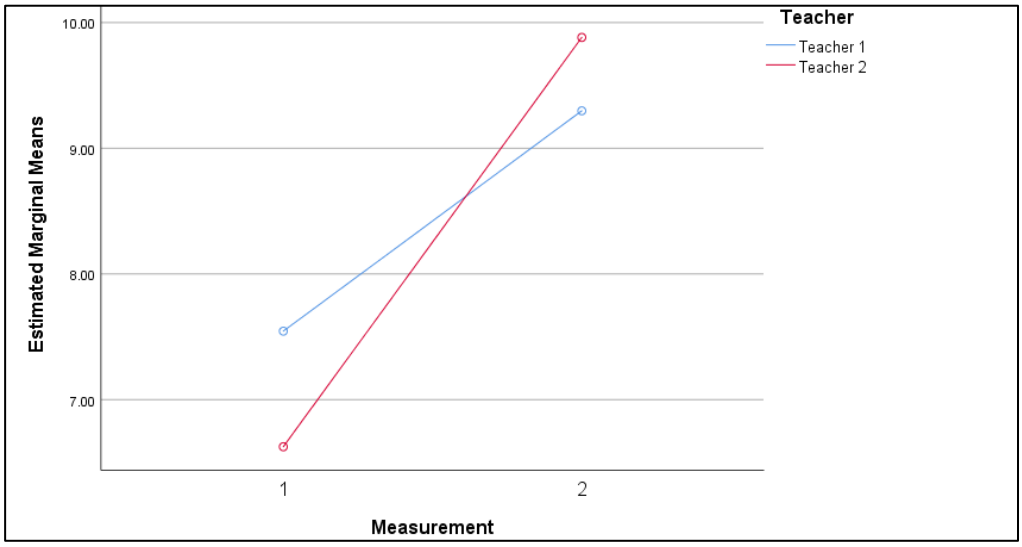


Figure F.3 Effect of Pretest Scores and Teachers' Attitude on Computer Programming Self-Efficacy Scale Posttest Scores

Results for SBPAT

Within Group Comparisons

Table F.9 Paired Samples t-Tests (by Teachers)

Pair	Test /Dimension (Group)	*N*	*M*	*SD*	*t*	*df*	*p*
Pair 1	Teacher 1: Experimental group (Pretest)	45	84.67	9.20	0.927	44	.36
	Teacher 1: Experimental group (Posttest)	45	83.56	9.45			
Pair 2	Teacher 2: Experimental group (Pretest)	45	83.33	11.87	-0.368	44	.72
	Teacher 2: Experimental group (Posttest)	45	83.89	11.07			
Pair 3	Teacher 1: Control group (Pretest)	47	69.04	15.13	**4.433**	**46**	**.00****
	Teacher 1: Control group (Posttest)	47	59.57	17.06			
Pair 4	Teacher 2: Control group (Pretest)	47	67.23	18.38	**2.834**	**46**	**.00****
	Teacher 2: Control group (Posttest)	47	61.38	19.47			

Between Group Comparisons

Table F.10 Independent Samples t-Tests (by Teachers)

Scale	Group	*N*	*M*	*SD*	*t*	*df*	*p*
SBPAT (Pretest)	Teacher 1	93	7.64	3.01	0.66	182	.51
	Teacher 2	95	6.58	2.67			
SBPAT (Posttest)	Teacher 1	89	9.30	2.35	-0.40	182	.70
	Teacher 2	93	9.88	2.61			

Covariance Analysis

Table F.11 Covariance Analysis of Computer Programming Self-Efficacy Scale Pretest and Posttest Scores (Students of Teacher 1 & Teacher 2)

Source	*SS*	*df*	*MS*	*F*	*p*	η_p^2
CPSES Pre-Test	36,733.543	1	36,733.543	235.073	**.000****	.565
Teacher	280.433	1	280.433	1.795	.182	.010
Error	28,283.848	181	156.264			

[a.] R Squared = .565 (Adjusted R Squared = .561)

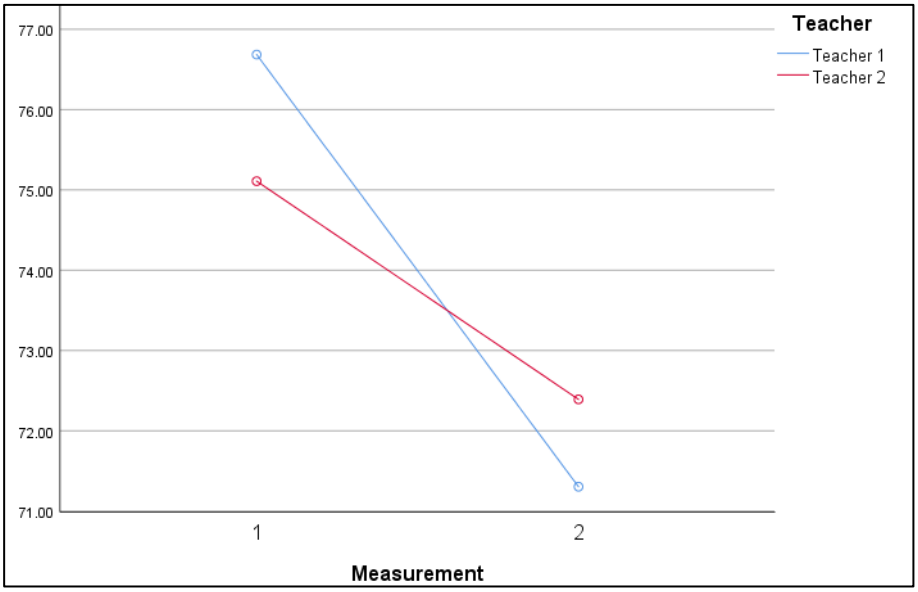
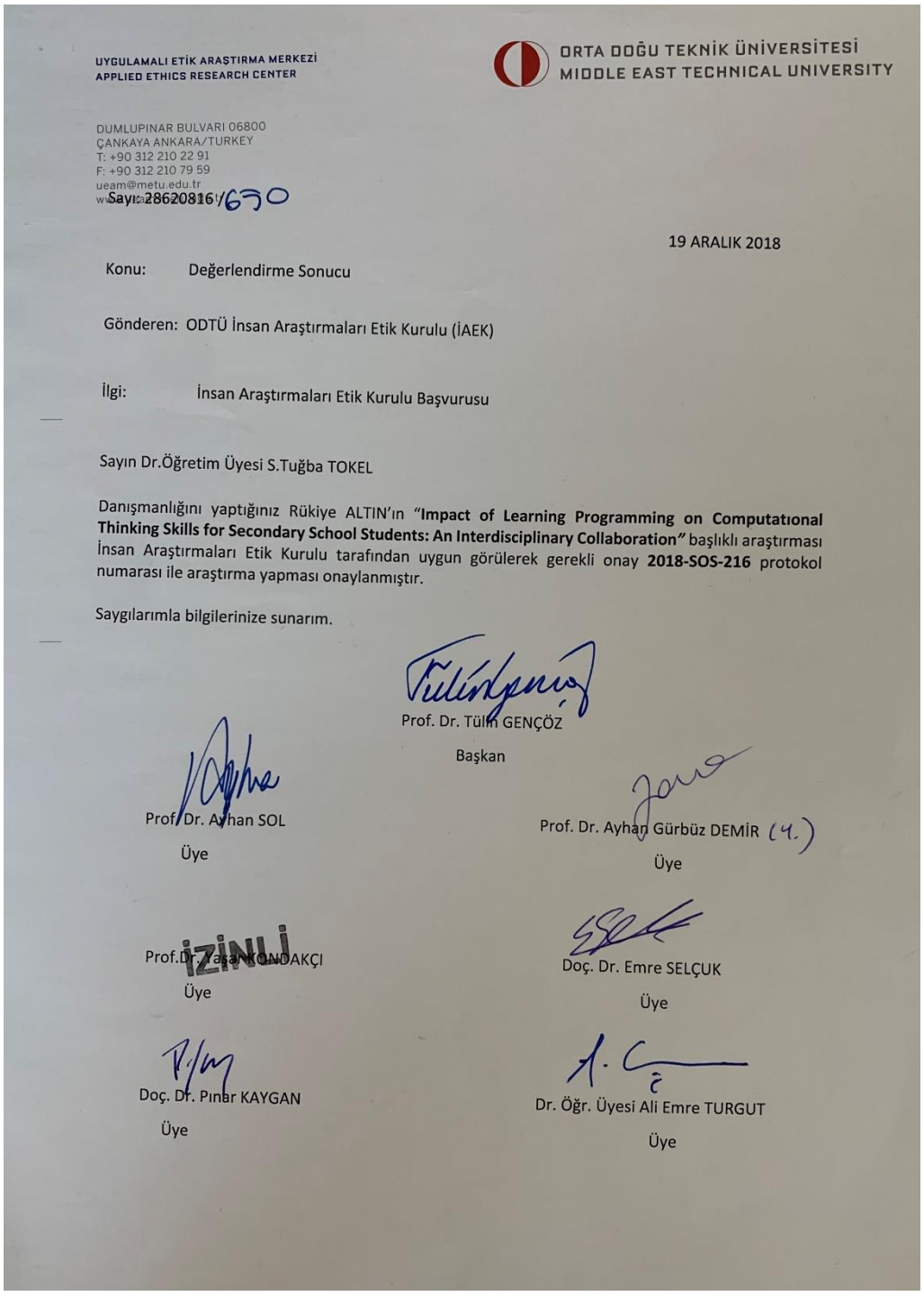


Figure F.4 Effect of Pretest Scores and Teachers' Attitude on Small Basic Programming Achievement Test's Posttest Scores

G. Permissions

Permission of METU -Ethical Committee



UYGULAMALI ETİK ARAŞTIRMA MERKEZİ
APPLIED ETHICS RESEARCH CENTER

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY

DUMLUPINAR BULVARI 06800
ÇANKAYA ANKARA/TURKEY
T: +90 312 210 22 91
F: +90 312 210 79 59
ueam@metu.edu.tr

Sayı: 28620816/690

19 ARALIK 2018

Konu: Değerlendirme Sonucu

Gönderen: ODTÜ İnsan Araştırmaları Etik Kurulu (İAEK)

İlgi: İnsan Araştırmaları Etik Kurulu Başvurusu

Sayın Dr.Öğretim Üyesi S.Tuğba TOKEL

Danışmanlığını yaptığınız Rükiye ALTIN'ın **"Impact of Learning Programming on Computatıonal Thinking Skills for Secondary School Students: An Interdisciplinary Collaboration"** başlıklı araştırması İnsan Araştırmaları Etik Kurulu tarafından uygun görülerek gerekli onay **2018-SOS-216** protokol numarası ile araştırma yapması onaylanmıştır.

Saygılarımla bilgilerinize sunarım.

Prof. Dr. Tülin GENÇÖZ
Başkan

Prof. Dr. Ayhan SOL
Üye

Prof. Dr. Ayhan Gürbüz DEMİR (4.)
Üye

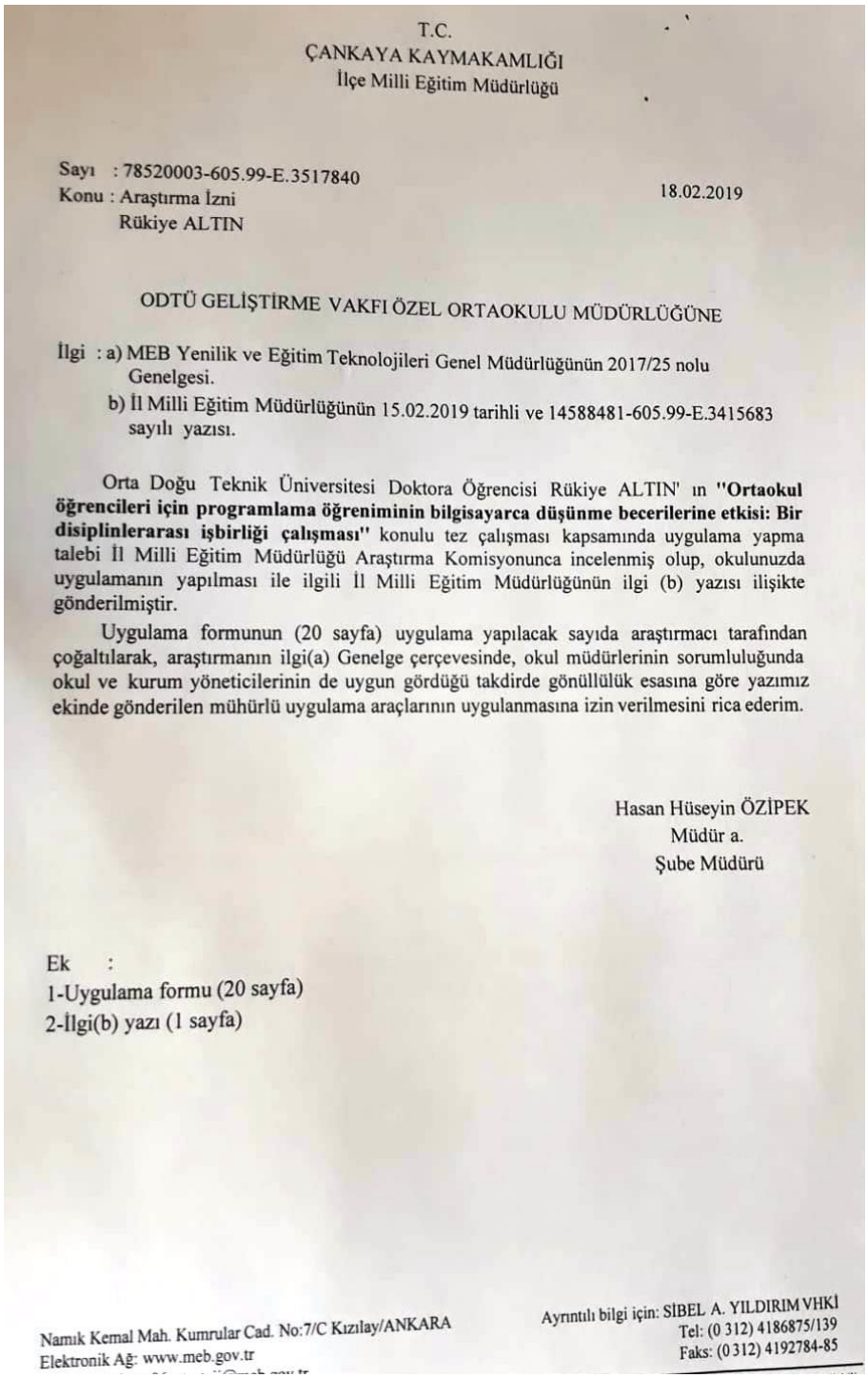
Prof.Dr. Zaza İZİNLİ KONDAKÇI
Üye

Doç. Dr. Emre SELÇUK
Üye

Doç. Dr. Pınar KAYGAN
Üye

Dr. Öğr. Üyesi Ali Emre TURGUT
Üye

Permission of Ministry of National Education



T.C.
ÇANKAYA KAYMAKAMLIĞI
İlçe Milli Eğitim Müdürlüğü

Sayı : 78520003-605.99-E.3517840
Konu : Araştırma İzni
Rükiye ALTIN

18.02.2019

ODTÜ GELİŞTİRME VAKFI ÖZEL ORTAOKULU MÜDÜRLÜĞÜNE

İlgi : a) MEB Yenilik ve Eğitim Teknolojileri Genel Müdürlüğünün 2017/25 nolu Genelgesi.
b) İl Milli Eğitim Müdürlüğünün 15.02.2019 tarihli ve 14588481-605.99-E.3415683 sayılı yazısı.

Orta Doğu Teknik Üniversitesi Doktora Öğrencisi Rükiye ALTIN' ın **"Ortaokul öğrencileri için programlama öğreniminin bilgisayarca düşünme becerilerine etkisi: Bir disiplinlerarası işbirliği çalışması"** konulu tez çalışması kapsamında uygulama yapma talebi İl Milli Eğitim Müdürlüğü Araştırma Komisyonunca incelenmiş olup, okulunuzda uygulamanın yapılması ile ilgili İl Milli Eğitim Müdürlüğünün ilgi (b) yazısı ilişikte gönderilmiştir.

Uygulama formunun (20 sayfa) uygulama yapılacak sayıda araştırmacı tarafından çoğaltılarak, araştırmanın ilgi(a) Genelge çerçevesinde, okul müdürlerinin sorumluluğunda okul ve kurum yöneticilerinin de uygun gördüğü takdirde gönüllülük esasına göre yazımız ekinde gönderilen mühürlü uygulama araçlarının uygulanmasına izin verilmesini rica ederim.

Hasan Hüseyin ÖZİPEK
Müdür a.
Şube Müdürü

Ek :
1-Uygulama formu (20 sayfa)
2-İlgi(b) yazı (1 sayfa)

Namık Kemal Mah. Kumrular Cad. No:7/C Kızılay/ANKARA
Elektronik Ağ: www.meb.gov.tr

Ayrıntılı bilgi için: SİBEL A. YILDIRIM VHKİ
Tel: (0 312) 4186875/139
Faks: (0 312) 4192784-85

ORTA DOĞU TEKNİK ÜNİVERSİTESİNE

25/12/2018

BAŞVURU NO	20181225553926547
ÜNİVERSİTE ADI	ORTA DOĞU TEKNİK ÜNİVERSİTESİ
ENSTİTÜ ADI	Fen Bilimleri Enstitüsü / Bilgisayar ve Öğretim Teknolojileri Eğitimi
BÖLÜM ADI	
ÜNVAN	Öğrenci
TC KİMLİK NUMARASI	35071917148
KONU	Impact of Learning Programming on Computational Thinking Skills for Secondary School Students: An Interdisciplinary Collaboration
ARAŞTIRMA TÜRÜ	Doktora Tezi
ÖRNEKLEM GRUBU	Öğrenci,
KAPSAMI	Okul/Kurum,
İLLER	ANKARA
KURUM TÜRLERİ	Özel Ortaokul,
İLETİSİM BİLGİLERİ	Adres:Üniversiteler Mahallesi, Dumlupınar Bulvarı No:1, 06800 ODTÜ Çankaya/Ankara- Telefon:(530) 315-9665- Eposta:altin.rukiye@metu.edu.tr

Yukarıda bilgileri bulunan proje uygulamarı için Milli Eğitim Bakanlığından gerekli izinlerin alınması hususunda gereğini bilgilerinize arz ederim.

Ek listesi

Tez Önerisi

Veli Onam Formu
Veri toplama araçları

İmza
RÜKİYE ALTIN
Öğrenci

Dilekçe ve eklerinin üst yazı ile ANKARA VALİLİĞİ İL MİLLİ EĞİTİM MÜDÜRLÜĞÜNE ulaştırılması gerekmektedir.

Consent Forms

Parent Information and Consent Form for Students

Sayın Veli,

Bu çalışma ODTÜ - Bilgisayar ve Öğretim Teknolojileri Eğitimi Bölümü öğretim üyelerinden Yrd. Doç. Dr. S. Tuğba TOKEL danışmanlığında, doktora öğrencisi Rükiye ALTIN tarafından yürütülmektedir. Bu doküman Bilişim Teknolojileri ve Yazılım dersinde yapılması planlanan araştırma konusu ve uygulaması hakkında sizleri bilgilendirmek için hazırlanmıştır. Çalışmanın amacı öğrencilere daha etkin bir biçimde programlamayı öğretmek, programlamayı öğretirken öğrencilerin bilgisayarca düşünme becerilerine (computational thinking skills) katkıda bulunmaktır.

Bu yüzden bu çalışma iki alana odaklanmaktadır. Öncelikle, bu çalışma geleneksel ve disiplinler arası öğretim yöntemleri ile öğretilen programlamanın öğrencilerin programlama becerileri üzerindeki etkisine odaklanmaktadır. İkinci olarak, çalışma disiplinler arası yaklaşım ve geleneksel öğrenme metotları ile öğretilen programlamanın, öğrencilerin bilişsel düşünme becerileri üzerine olan etkisini kapsamaktadır.

Small Basic programlama dili müfredat kapsamında olup, öğrencilerimize her yıl öğrettiğimiz bir konu. Bu araştırmada da müfredat dışına çıkılmadan Small Basic ile programlama konusu ele alınacaktır. Araştırmanın birinci aşamasında deneysel ve kontrol grupları oluşturulacak. Deneysel gruba matematik işbirliği ile Small Basic programı anlatılırken, kontrol grubuna klasik programlama anlatım yöndemi ile Small Basic programı öğretilecek. Her iki gruba da Bilişim Teknolojileri ve Yazılım dersi kazanımları aynı verilirken, deneysel gruba buna ek olarak matematik kazanımları dahil edilecek. Araştırmanın ikinci bölümünde çıkan farka göre hangi yöntem ile programla anlatımının öğrencilerin anlama açıcından daha kolay olduğu, bilişsel düşünme becerilerine katkısı ve programlama yeteneklerine olan etkisi tartışılacak.

Bu çalışma hakkında daha fazla bilgi almak, her türlü soru ve/veya yorumlarınız için Rükiye ALTIN . altin.rukiye@metu.edu.tr e-posta adresinden iletişime geçebilirsiniz.

Lütfen bu araştırmaya katılmak konusundaki tercihinizi aşağıdaki seçeneklerden size en uygun gelenin altına imzanızı atarak belirtiniz.

A) Velisi bulunduğum .....'nın bu araştırmaya katılımcı olmasına **izin veriyorum**. Çocuğumun çalışmayı istediği zaman yarıda kesip bırakabileceğini biliyorum ve verdiği bilgilerin bilimsel amaçlı olarak kullanılmasını kabul ediyorum.

Velinin Adı-Soyadı .....

İmza .....

B) Velisi bulunduğum .....'nın bu çalışmaya **katılmasını kabul etmiyorum** ve **katılımcı olmasına izin vermiyorum**.

Velinin Adı-Soyadı .....

İmza .....

H. Post-Research Information Form

Araştırma Sonrası Bilgilendirme Formu

Öncelikle araştımamıza katıldığınız için teşekkür ederiz. Katıldığınız araştırmanın amacı öğrencilere daha etkin ve etkili bir şekilde programlamayı öğretebilmek için klasik yöntem ve disiplinlier arası işbirliği yaklaşımını karşılaştırmaktır. Literatüre göre programlama öğretimi ve öğrenimi, diğer derslerle bağdaştırıldığında daha kolay olmaktadır. Bu yüzden bu çalışma geleneksel ve disiplinler arası öğretim yöntemleri ile öğretilen programlamanın öğrencilerin programlama becerileri üzerindeki etkisine odaklanmaktadır. İkinci olarak, çalışma disiplinler arası yaklaşım ve geleneksel öğrenme metotları ile öğretilen programlamanın, öğrencilerin bilişsel düşünme becerileri üzerine olan etkisini kapsamaktadır.

Araştırmanın birinci aşamasında deneysel ve kontrol grupları oluşturuldu. Her iki gruba da Matematik Tutum Ölçeği, Bilgi İşlemsel Düşünmeye Yönelik Öz Yeterlilik Algısı Ölçeği ve Programlamaya İlişkin Öz Yeterlilik Algısı Ölçeği ön test olarak uygulanmıştır. Uygulama sonrasında, algoritma ve Small Basic kullanımları yine her iki gruba aynı ders planı ile anlatılmışır. Derslerin devamında ise deneysel gruba matematik iş birliği ile programlama anlatılırken, kontrol gruba klasik yöntem ile programlama anlatılmıştır. Her iki gruba da Bilişim Teknolojileri ve Yazılım kazanımları aynı verilirken, deneysel gruba buna ek olarak matematik kazanımları eklenmiştir.

Ders anlatımlarının sonunda iki gruba ön test olarak uygulanan Matematik Tutum, Bilgi İşlemsel Düşünmeye Yönelik Öz Yeterlilik Algısı ve Programlamaya İlişkin Öz Yeterlilik Algısı ölçekleri son test olarak uygulanmıştır. Bunun yanı sıra Small Basic Kazanım testi iki gruba da uygulanıp, hangi grup kazanımları daha iyi öğrendi kıyaslaması yapılmıştır. Son rasgele seçilen öğrenci ve dersi anlatan öğretmenlerle değerlendirme için görüşme yapılıp, sorular yönlendirilmiştir.

Eğer araştırmayla ilgili sorularınız varsa Rükiye ALTIN'a altin.rukiye@metu.edu.tr e-posta adresinden iletişime geçerek sorabilirsiniz.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Altın, Rükiye
Nationality: Turkish (TC)
email: altin.rukiye@metu.edu.tr, altin.rukiye@gmail.com

EDUCATION

Degree	Institution	Year of Graduation
MA	Bilkent University Comp. Edu. and Inst. Tech.	2011
BA	Bilkent University Comp. Edu. and Inst. Tech.	2010
High School	Bahçelievler Deneme High School, Ankara	2003

WORK EXPERIENCE

Year	Place	Enrollment
2011 – present	ODTÜ Geliştirme Vakfı Okulları	ICT Teacher

FOREIGN LANGUAGES

Advanced English

PUBLICATIONS

1. Altin, R., Tokel, T., & Delialioglu, O. (2020). The Effects of Mathematics on Programming Skills and its Retention: An Experimental Study. Manuscript submitted for publication.
2. Altin, R., Tokel, T., & Delialioglu O. (2020). *How to Improve Computational Thinking Skills with Computer Science Integration: An Experimental Study*. Manuscript submitted for publication.
3. Altin, R., & Erkan Isler, E. (2020). Integration of ICT in EFL classes by using digital storytelling to discover common moral values: An international project. In D. Schmidt-Crawford (Ed.), Proceedings of Society for Information Technology & Teacher Education International Conference (pp. 148-153). Association for the Advancement of Computing in Education (AACE). https://www.learntechlib.org/primary/p/215744/

Altin, R., & Erkan İsler, E. (2019). Building Bridges between ICT and English: A Blended Learning Example with International Projects Integration.

4. Altin, R., Bektik, M., Eksioglu, N., Koray, C., Canbek Öner, Ö., Sadetaş, M., Şener, H., Şimşek, D., Ma, C.-C., Price, C., & Routh, C. (2009). Working across time zones in cross-cultural student teams. In ITiCSE '09: Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education (pp. 360). https://doi.org/10.1145/1562877.1563002
5. Altin, R., Bektik, M., Ekşioğlu, N., Koray, C., Canbek Öner, Ö., Sadetaş, M., Şener, H., Şimşek, D., Ma, C.-C., Price, C., & Routh, C. R. (2009). Use of intuitive tools to enhance student learning and user experience. In ITiCSE '09 Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education (pp. 365). https://doi.org/10.1145/1562877.1563007

AWARDS

2019- Enthusiastic Researcher Award – A Blended Learning Example with International Projects Integration! Singapore.