Instructional Technology and Lifelong Learning Vol. 5, Issue 1, 21-46 (2024)

https://dergipark.org.tr/en/pub/itall

ITALL

Research Article

ISSN: 2717-8307

The Effect of Math-Supported Introductory Programming Education on Computational Thinking

Muradiye Bozal[1], Polat Şendurur *[2]

* *Corresponding Author, polat.sendurur@omu.edu.tr*
[1]*Ministry of National Education, Türkiye*
[2]*Ondokuz Mayıs University, Faculty of Education, Türkiye*

Abstract

The study aims to research the effect of an introductory programming course with math-based programming activities on computational thinking skills and self-efficacy. A static-groups pre-test post-test quasi-experimental design was used. One hundred and seventy-six 6th-grade public school students participated in the study. Eighty-nine students were in the experimental group, and 87 were in the control group. While the students in the experimental group received introductory programming education with Math-supported activities, the students in the control group received programming education with traditional course activities. Equivalent programming activities were carried out in both groups. Data were collected via the Computational Thinking Test and Self-Efficacy Perception Scale for Computational Thinking Skills. After the study, post-test scores were analyzed using ANCOVA analysis by controlling pre-test scores. The findings indicated no difference between the two groups regarding computational thinking test performance. Similarly, no conclusion stated a difference between the groups' perceptions of self-efficacy of computational thinking. According to these results, evaluations regarding the positive and negative effects of using mathematics and programming together in an elementary programming education, which is thought to be related to Computational Thinking Skills, were reached at the skills of the study.

Keywords: Math, Computational Thinking, Self-efficacy, Programming

Citation: Bozal, M., & Sendurur, P (2024). The effect of Math-supported introductory programming education on computational thinking. *Instructional Technology and Lifelong Learning*, *5*(1), 21- 46. https://doi.org/ 10.52911/itall.1394556

Matematik Destekli Programlamaya Giriş Eğitiminin Bilgi İşlemsel Düşünme Üzerindeki Etkisi

Özet

Bu çalışmanın amacı, matematik temelli programlama etkinlikleri ile temel programlama eğitiminin bilgi işlemsel düşünme becerileri ve öz yeterlilik üzerindeki etkisini araştırmaktır. Çalışmada statik gruplar ön-test son-test yarı deneysel desen kullanılmıştır. Çalışmaya bir devlet okulunda öğrenimine devam eden yüz yetmiş altı 6. sınıf öğrencisi katılmıştır. Bu öğrencilerin 89'u deney grubunda, 87'si ise kontrol grubunda yer almıştır. Deney grubundaki öğrenciler matematik destekli etkinliklerle programlamaya giriş eğitimi alırken, kontrol grubundaki öğrenciler geleneksel ders etkinlikleri ile programlama eğitimi almışlardır. Her iki grupta da eşdeğer programlama etkinlikleri gerçekleştirilmiştir. Veriler Bilgi İşlemsel Düşünme Testi ve Bilgi İşlemsel Düşünme Becerilerine Yönelik Öz Yeterlik Algısı Ölçeği aracılığıyla toplanmıştır. Çalışma sonunda, son test puanları ön test puanları kontrol edilerek ANCOVA ile analiz edilmiştir. Bulgular, iki grup arasında bilgi işlemsel düşünme performansı açısından bir fark olmadığını göstermiştir. Benzer şekilde, grupların bilgi işlemsel düşünme öz yeterlilik algıları arasında da bir fark olmadığı sonucuna varılmıştır. Bu sonuçlara göre, ilköğretim programlama eğitiminde matematik ve programlamanın birlikte kullanılmasının Bilgi İşlemsel Düşünme Becerileri ile ilişkili olabileceği düşünülen olumlu ve olumsuz etkilerine ilişkin değerlendirmelere ulaşılmıştır.

Anahtar Kelimeler: Matematik, Bilgi işlemsel düşünme, Öz- yeterlik, Programlama

Date of Submission	22.11.2023
Date of Acceptance	05.02.2024
Date of Publication	30.06.2024
Peer-Review	Double anonymized - Two External
Ethical Statement	It is declared that scientific and ethical principles have been followed while carrying out and writing this study and that all the sources used have been properly cited.
Acknowledgements	This article is the revised and developed version of the unpublished conference presentation entitled "Matematik destekli temel programlama eğitiminin bilgi işlemsel düşünme becerisi üzerindeki etkisi", orally delivered at the International Conference on Educational Technology and Online Learning (2022). This article is extracted from my master thesis "Effect of math supported introductory programing education on computational thinking", supervised by Polat Şendurur (Master's Thesis/Ondokuz Mayıs University, Samsun, Türkiye, 2022).
Author(s) Contribution	**Bozal:** Data curation, Writing-Original draft preparation, Conceptualization. **Sendurur:** Methodology, Writing- Reviewing and Editing.
Plagiarism Checks	Yes - Turnitin
Conflicts of Interest	The author(s) has no conflict of interest to declare.
Complaints	ithalljournal@gmail.com.
Grant Support	The author(s) acknowledge that they received no external funding in support of this research.
Copyright & License	Authors publishing with the journal retain the copyright to their work licensed under the **CC BY 4.0**.

1. Introduction

Technology continues to develop rapidly and occupies more space in all areas of life. Developed countries invest more in studies in the field of technology because economic growth is parallel to technological development. They integrate computer science subjects into their curricula, starting from primary education, to increase the pace of their technological development. Countries such as England, France, Finland, Poland, Denmark, and Türkiye have recently updated their curricula, including computational thinking (CT) skills, algorithms, and coding. In Turkey, the curriculum of some courses has been rearranged, and content has been produced in recent years to improve CT skills (Gülbahar & Kalelioğlu, 2018). As a result of the rapid development of computer science and technology, the development of students' competence in solving technological problems is considered an important issue in education systems. Problems that arise in many fields, such as education, health, trade, industry, transportation, and entertainment, are solved thanks to the developed hardware and software. According to Wing (2006), solutions developed to manage our daily lives, communicate, and interact with others should not be limited to only physical software and hardware products. The main point to focus on is ensuring students gain computational concepts. The International Society for Technology in Education (ISTE) aims to contribute to educational institutions' planned use of technology and be a guide. For this reason, it develops some standards for students, teachers, administrators, coaches, and computer educators. The standards developed by the institution for students (ISTE, 2016) include (i) Empowered Learner, (ii) Digital Citizenship, (iii) Knowledge Builder, (iv) Innovative Designer, (v) Computational Thinker, (vi) Creative Communicator, and (vii) Global Collaborator has defined its features.

These standards also show that computational thinking is one of the essential skills individuals in the 21st century should acquire. The importance of computational thinking has raised questions such as "What is computational thinking?", "How is it taught?" and "How is it measured?" According to Wing (2008), computational thinking requires thinking like an engineer in addition to mathematical thinking when solving problems. Papert (1980) observed that his students' thinking abilities were significantly different when writing computer programs and that they developed their cognitive abilities. ISTE (2016) stated that computational thinking includes creative thinking, algorithmic thinking, problem-solving, and collaborative learning skills. These statements show that computational thinking is a skill

created by combining the gains of subjects such as mathematics, engineering, and computer programming. Many countries think in this direction and equip teaching programs with relevant topics to enable students to think like engineers and acquire these skills. In our country, we can also find examples of computer and non-computer coding and robotics studies starting from the primary level. For example, in the 5th and 6th grades of middle school, the example of the Algorithms and Basic Programming education in the Information Technologies and Software course can be given for the relevant situation. Scratch is generally preferred in these training carried out in block-based programming environments. In this regard, studies showing that Scratch is the primary tool and can be effective in computational thinking are available in the literature (Oluk et al., 2018; Şimşek, 2018)

However, thinking that computational thinking can only be learned by programming reflects a limited perspective. Computational thinking does not only consist of writing code in a computer environment; it is gained during the process of understanding the problem, analyzing, abstracting, algorithmic thinking, and creating flow diagrams before writing the program. Disciplines that apply problem-solving steps have the potential to develop computational thinking. Studies show that subjects such as Biology, Physics, Mathematics, and English develop computational thinking skills (Lockwood & Mooney, 2017).

From another perspective, the limitations of the underlying information processing device force computer scientists to think not only mathematically but also numerically (Wing, 2006). In other words, there is a mutually beneficial relationship between computational thinking and mathematics. A study by Sung and Black (2020) observed that when students worked on math problems thinking like a computer programmer, their task analysis, sequential thinking, procedural thinking, and coding skills improved. A similar result was obtained in a study conducted by Rodríguez-Martínez et al. (2020), and they showed that programming activities contributed relative to the adequacy of solving some math problems. In conclusion, it can be inferred that integrating mathematics gains with computer science concepts may impact computational thinking skills.

This research investigates the effects of mathematics-supported introductory programming education in the Information Technologies and Software course on the computational thinking test performance and self-efficacy of 6th-grade students. By presenting the gains of information

technologies and mathematics courses that effectively teach computational thinking skills in the same activity, the role of mathematics-supported activities in developing computational thinking skills in Scratch-based basic programming education is focused. Another aim is to focus on the role of mathematics-supported programming education in developing students' computational thinking skills and making inferences about the direction of interdisciplinary studies. Additionally, this study aims to provide examples of research in the fields of programming, computational thinking, and mathematics in terms of content, method, duration, activities, and implementers. In this context, the following research questions have been attempted to be answered:

(1) Does mathematics-supported basic programming education affect the computational thinking test performance of 6th-grade students?

(2) Does mathematics-supported basic programming education affect the self-efficacy perception of computational thinking of 6th-grade students?

2. Theoretical Framework

2.1. Computational Thinking

Although computational thinking has become quite popular recently, the concept dates back a few decades. Studies on the logic behind computer functioning as a problem-solving method were first initiated by Alan Perlis in the 1960s (Özçınar, 2017). The term computational thinking was first used by Papert (1996) in 1996. However, much earlier, Papert (1980) observed that the thinking abilities of his students significantly differed when they wrote computer programs, and this situation also developed their cognitive abilities. He stated that children could develop computational thinking skills by learning the LOGO programming language. This idea emerged from recognizing the development of thinking skills that occurred during the process of students' programming. The concept of computational thinking was first included in Jeannette M. Wing's study in 2006. She stated that computer science is not limited to computer programming and that thinking like a computer scientist requires thinking at multiple abstraction levels. According to Wing (2008), computational thinking is a type of analytical thinking, and the general ways of solving a problem rely on mathematical, engineering, and scientific thinking skills.

The literature provides many definitions of computational thinking. For example, Wing (2006) expresses computational thinking as problem-solving, system design, and understanding human behavior using basic computer science concepts. According to Aho (2012), it contains thinking processes in which problem-solving solutions can be presented in steps and algorithms compatible with computer logic. Syslo and Kwiatkowska (2013) defined it as mental activity in formulating a problem. Korkmaz et al. (2015) expressed it as a method of problem-solving, system design, drawing attention to the basic concepts of computer science, and understanding human behavior. Curzon (2015) attempted to explain computational thinking as a problem-solving ability for humans. According to Angeli et al. (2016), it is a thought process that uses the elements of abstraction. According to Şahiner and Kert (2016), it is a comprehensive skill that includes critical thinking, problem-solving, algorithmic thinking, and adapting the working style of the computer to daily life. In a joint statement by the International Society for Technology in Education (ISTE) and the Computer Science Teachers Association (CSTA), an operational definition is proposed for the ability to solve computational problems using computer assistance. The definition includes skills such as formulating, organizing, and presenting data, algorithmic thinking, transfer, and generalization. Therefore, reaching a clear definition of computational thinking in national and international literature may not be possible. However, definitions are expressed with problem-solving, algorithms, abstraction, and critical thinking concepts.

2.2. Subcomponents of computational thinking

The subcomponents of computational thinking are also characterized by varying opinions, just as its name and definition are. For example, Wing (2006) proposed that computational thinking includes problem-solving, abstraction, decomposition, intuitive thinking, and mathematical and engineering-based thinking. The BBC's education website in the UK includes a guide to computational thinking that includes decomposition, abstraction, pattern recognition, and algorithms. ISTE (2016) listed the subcomponents of computational thinking: data collection, data analysis, data presentation, decomposition, abstraction, algorithms, automation, testing, parallelism, and simulation. Tosik Gün and Güyer (2019) systematically reviewed the literature on computational thinking. The study stated that the most commonly accepted components in evaluating computational thinking are abstraction, algorithmic thinking, decomposition, testing and debugging, data literacy, sorting, and flow control structures. According to the

literature review conducted by Gulbahar and Kalelioglu (2015), the most frequently encountered subcomponents of computational thinking are understanding problem, decomposition, pattern finding/recognition, abstraction, algorithms, testing/debugging, automation, data collection/analysis, and modeling.

The subcomponents of computational thinking have been influential in the measurement of this skill, and researchers have developed various tools to measure it (Dolmacı & Akhan, 2020; Gülbahar & Kalelioğlu, 2018; Korkmaz et al., 2015; Kukul & Karatas, 2019; Özmen, 2016). Similar dimensions are found in the scales developed for different participant groups (Tosik et al., 2019). Therefore, the most commonly seen subcomponents in the scales are abstraction, algorithmic thinking, decomposition, testing and debugging, and data literacy (Tosik et al., 2019).

2.3. Developing Computational Thinking and Scratch

According to Weinberg (2013), there are four different approaches to developing computational thinking: CS Unplugged, programming tools, game or robot programming, and interdisciplinary applications. In addition to tools, some strategies are also used to develop computational thinking. Hsu et al. (2018) have summarized the strategies used to impart computational thinking skills to students. Some teaching approaches for computational thinking are problem-based, collaborative, project-based, game-based, scaffolding, storytelling, computational learning theory, aesthetic experience, concept-based learning, object-oriented learning, human-computer interaction-based learning instruction, and universal design for learning. When these tools and approaches are considered, the Scratch block-based programming environment has emerged in terms of availability and usability. One of the crucial reasons for its emergence is that it works with the drag and drop logic and allows even people without programming knowledge to use it (Resnick et al., 2009) easily, and thus addresses users from the lowest level to the highest level (Grover & Pea, 2013). In this perspective, several scratch-based programming environments and studies targeting computational have been conducted (Adsay et al., 2020; Ataman-Uslu et al., 2018; Oluk et al., 2018; Vatansever, 2018; Yünkül et al., 2017).

2.4. The relationship among Math, Scratch, and computational thinking

The standards ISTE (2016) put forth do not limit computational thinking to just computer science. In addition to computer science, computational thinking is associated with mathematics, science and technology, social studies, and language. Lockwood and Mooney (2017) mention that many studies show that computational thinking skills can be integrated into Biology, Physics, Math, and English courses.

Mathematics and computational thinking are closely related to the analysis and interpretation of data and the communication of information. Mathematical methods, data collection and analysis tools, and visualizations provide an ease for students to work with large amounts of data. Furthermore, when students struggle to express findings in text or speech, they use mathematical representations, data visualizations, simulations, and graphic representations (Wilkerson & Fenwick, 2017). These sub-tools used in mathematics align with the sub-components of computational thinking and show the connection between the two fields.

Many studies in the literature have investigated the development of computational thinking skills through mathematical activities using Scratch. In their study, Sung et al. (2017) aimed to provide students with computational thinking skills through various levels of concretized activities, and the results showed that activities supported by computational thinking improved students' mathematical understanding and programming skills in Scratch Jr. Another study by Okuducu (2020) examined the effect of using Scratch on students' academic achievement and attitude towards algebra and found that Scratch-based lesson activities created a positive difference in their algebraic expression achievement and attitude. In another study, it was found that a majority of students' learning difficulties in mathematics were addressed with mathematical games designed with the Scratch programming tool (Çubukluöz, 2019).

Overall, research suggests that using Scratch programming or coding tools with Mathematics has an advantage. It has been concluded that when Mathematics is taught with Scratch, it is more successful than traditional methods (Çubukluöz, 2019; Okuducu, 2020). The fact that sub-operations such as decomposition, abstraction, modeling, simulation, and pattern recognition used in programming lessons are also frequently used in mathematics lessons may facilitate the understanding of mathematical subjects. Based on the existing literature and various

experiences, it is thought that programming education with mathematical content has the potential to impact computational thinking.

3. Method

The study used a static group's pre-test, post-test week-experimental design. Since the initial states of the groups are crucial to understanding the effect of the manipulation of the independent variable on dependent variables, researchers included similar static groups in the study to ensure that the groups were close to each other before the experiment. However, since the groups could not be formed by the researcher, the study was continued with a weak experimental design (Fraenkel et al., 2012). In the data collection process, the students were first given a computational thinking test and a self-perception of computational thinking scale as pre-tests. Then, the experimental group was given four weeks of programming education with mathematical support, and the training was supported by various examples. The mathematical support examples include mathematics topics covered in the 6th grade Math course curriculum and taught in the first month of the first semester. During the same period, various in-class activities based on the Information Technologies and Software course were applied to the control group according to the teaching program. At the end of the process, the same measurement tools were applied as post-tests.

3.1. Participants

The convenience sampling method was used in the study. According to Cresswell (2012), convenience sampling enables the study with participants who are willing to participate and available for the study. After applying convenience sampling methods, the study group consisted of 200 students in the 6th grade at a middle school in Samsun in the 2021-2022 academic year. The data of 176 of these students who had full pre-test and post-test data and students were used in the analyses. There were 89 students in the experimental group: 46 girls and 43 boys. In the control group, 87 students, 47 girls, and 40 boys participated in the study.

Before participating in the study, the students had attended the Information Technologies and Software course in the 5th grade in a way consistent with the national instructional program during the first and second semesters online due to the pandemic conditions. In the second semester of the 5th grade, activities on introductory programming topics were carried out in

the code.org and Scratch environments. In this context, participant students have prior knowledge of programming.

3.2. Instruments

Computational Thinking Test (CTT) and Computational Thinking Self-Efficacy Scale (CTSES) were used as data collection tools. The tool used to measure the students' computational thinking skill levels before and after the application was developed by Román-González et al. (2017) and adapted to Turkish by Çetin et al. (2020). The other measurement tool is CTSES, which was developed by Gülbahar et al. (2019).

CTT

CTT is a 7-section, 28-item test developed by Román-González et al. (2017) that contains computer-based coding and visual coding tools. It is designed to determine computational thinking level in the context of programming and coding. The test measures the ability to solve problems and formulate equations using fundamental concepts such as loops, conditional structures, variables, arrays, and functions in programming languages. The original form of the test was developed for students in grades 7 and 8 (ages 12-14), but the developers have stated that it can also be used for students in grades 5-6 and 9-10. Table 3.2 shows the sections and number of questions in the test. When the scale is examined, it is seen that there are 28 questions in total, including basic sorting (4 questions), loops with a specific number of repetitions (4 questions), loops until a condition is met (4 questions), simple conditional statements (4 questions), complex conditional (if-else) statements (4 questions), loops that work only when a condition is true (4 questions), and simple functions (4 questions).

CTSES

Another measurement tool used in the study is CTSES, which was developed by Gülbahar et al. (2019). This scale consists of 5 factors and 36 items: algorithm design (9 items), problem-solving (11 items), data processing (7 items), basic programming (6 items), and self-confidence (6 items). The reliability coefficients of the scale range from 0.76 to 0.93 for each dimension. These values provide sufficient evidence for the reliability of the scale.

3.3. Procedure

The activities used in the experimental group in the study were prepared in collaboration with Mathematics teachers. Four activities were administered to the students in the experimental group. These activities were (i) operator precedence in natural numbers, (ii) calculating exponents, (iii) divisibility rules, and (iv) prime numbers. These activities were accompanied by the same programming topics in the control group, which included (i) a capital city game, (ii) an apple-picking game, (iii) an English word game, and (iv) a horoscope game. All activities were planned for 80 minutes and were carried out in the Scratch environment. The features of the program that would be produced and the rules it would have to meet were shared with the students before the activities, and necessary instructions were provided.

In the "operator precedence in natural numbers" activity, a game design activity was carried out in the Scratch environment to order operations according to the operator precedence rule. The activity aimed for students to learn to create a new character, make their character speak and change their appearance, hide and show, use the send and receive commands, use variables, generate random numbers, and perform operations using loops and conditional structures. In the "calculating exponents" activity, students were asked to write a program in the Scratch environment that calculates the exponent of a number entered by the user as many times as the user enters and displays the result on the screen. The activity aimed to develop the students' algorithmic thinking, conditional statements, loops, variables, and mathematical operation skills. In the use of the "divisibility rules" activity, the students were asked to write a program that determines whether one of the entered numbers is exactly divisible by the other. This application prioritized the conditional structure, array concept, and the MOD command, a mathematical function. In addition, text concatenation was emphasized. The last activity of the experimental group was the "prime numbers" activity. In this activity, the students designed a program to determine whether a number is a prime number and display the result on the screen. The topics of variables, conditional structure, loop structure, "or" statement, MOD command, and text concatenation were discussed in the activity. The control group activities include the same programming topics in the same order as the activities in the experimental group. The only difference between the activity groups is that the programming concepts are presented in the experimental group by matching them with the contents of the mathematics course.

3.4. Data analysis

In order to determine whether there is a difference between the groups in the sub-dimensions of each scale, analyses of variance (ANCOVA) were applied. Each ANCOVA was applied for each sub-dimension, and the pre-test results were included as a control variable in the analysis. In this way, the effect of the students' differences in the relevant dimension on the results at the beginning of the study was controlled. Before applying ANCOVA for each dimension, the assumptions required for this analysis were tested. These assumptions include the continuity of dependent, independent, and control, the groups being measured independently, the limited extreme values, and the residual values (residuals) normally distributed in each category context. When the relevant situations are examined, it is seen that all assumptions are met. In addition, the homogeneous distribution of variances, the linear correlation between the control variable and the dependent variable, the linearity of the regression lines, the homoscedasticity condition, and the normal distributions are also among the controlled assumptions. The assumption analysis of ANCOVA obtained in the study indicated that the collected data are appropriate for analysis.

4. Result

4.1. Computational Thinking Test

In this section, the results of each sub-dimension of the CTT are presented. The pre-test and post-test responses of the experimental and control groups to the CTT were analyzed using ANCOVA. Based on the results obtained, whether there was development in BID skill was interpreted.

Table 1.

ANCOVA results for "Basic Sorting."

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	26.741a	2	13.37	19.05	.00	.18
Intercept	77.466	1	77.47	110.38	.00	.39
Pre-Test	25.950	1	25.95	36.97	.00	.18
Group	2.387	1	2.39	3.40	.07	.02
Error	121.418	173	.70			
Total	1626.000	176				
Corrected Total	148.159	175				

Table 1 summarizes the ANCOVA results for the Basic Sorting dimension. In the ANCOVA analysis, where the post-test scores of the CTT-Basic Sorting dimension were the dependent variable, and the pre-test results were included as the control variable, no significant difference was found between the experimental and control groups. Although the difference is not significant (F(1.176)=19.05, p=.07), the participants in the experimental group (M=2.782) scored lower than the control group (M=3.017).

Table 2.

ANCOVA results for the dimension of "Loops"

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	61.629a	2	30.81	35.47	.00	.29
Intercept	39.432	1	39.43	45.39	.00	.21
Pre-Test	61.164	1	61.16	70.41	.00	.29
Group	.354	1	.35	.41	.52	.00
Error	150.280	173	.87			
Total	1292.000	176				
Corrected Total	211.909	175				

Table 2 summarizes the results of ANCOVA in the dimension of "Loops ."ANCOVA did not show a significant difference between the experimental and control groups. Although participants in the experimental group (M=2.522) obtained higher scores than control group students (M=2.432) when controlling for pre-test results, this difference was not significant (F(1.176)=35.47, p=.52).

Table 3.

ANCOVA results for the "Loops Until Condition Is Met"

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	32.562a	2	16.28	16.38	.00	.16
Intercept	47.866	1	47.87	48.15	.00	.22
Pre-Test	31.477	1	31.48	31.67	.00	.15
Group	.466	1	.47	.47	.49	.00
Error	171.984	173	.99			
Total	974.000	176				
Corrected Total	204.545	175				

Table 3 summarizes the ANCOVA results for the "Loops Until Condition Is Met" dimension. The ANCOVA analysis, in which the CTT "Loops Until Condition Is Met" test scores were the

dependent variable and pre-test scores were the control variable, did not find a significant difference between the experimental and control groups. Therefore, although participants in the experimental group (M=2.142) scored higher than students in the control group (M=2.039), this difference was not significant (F(1.176)=16.38, p=.49).

Table 4.

ANCOVA results for "Simple Conditional Statements"

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	14.258a	2	7.13	5.80	.00	.06
Intercept	108.784	1	108.78	88.46	.00	.34
Pre-Test	14.154	1	14.15	11.51	.00	.06
Group	.451	1	.45	.37	.55	.00
Error	212.737	173	1.23			
Total	695.000	176				
Corrected Total	226.994	175				

Table 4 summarizes the ANCOVA results for the "Simple Conditional Statements" dimension. The ANCOVA analysis, in which the CTT "Simple Conditional Statements" post-test scores were the dependent variable and pre-test scores were the control variable, did not find a significant difference between the experimental and control groups. Therefore, participants in the experimental group (M=1.580) scored lower than students in the control group (M=1.682), but this difference was not significant (F(1.176)=5.80, p=.55).

Table 5.

ANCOVA results for the "Complex Conditional Statements"

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	22.761a	2	11.38	9.96	.00	.10
Intercept	83.546	1	83.55	73.10	.00	.30
Pre-Test	20.814	1	20.81	18.21	.00	.09
Group	1.800	1	1.80	1.57	.21	.01
Error	197.733	173	1.14			
Total	763.000	176				
Corrected Total	220.494	175				

Table 5 summarizes the ANCOVA results for the "Complex Conditional Statements" dimension. The ANCOVA analysis, in which the CTT "Complex Conditional Statements" post-test scores were the dependent variable and pre-test scores were the control variable, did not find a significant difference between the experimental and control groups. Therefore,

participants in the experimental group (M=1.656) scored lower than students in the control group (M=1.858), but this difference was not significant (F(1.176)=9.96, p=.21).

Table 6.

ANCOVA results for the "Loops While Condition Is True"

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	11.018a	2	5.51	6.32	.00	.07
Intercept	55.440	1	55.44	63.56	.00	.27
Pre-Test	10.563	1	10.56	12.11	.00	.06
Group	.402	1	.40	.46	.50	.00
Error	150.891	173	.87			
Total	546.000	176				
Corrected Total	161.909	175				

Table 6 summarizes the ANCOVA results for the "Loops While Condition Is True" dimension. The ANCOVA analysis, in which the "Loops While Condition Is True" post-test scores were the dependent variable and pre-test scores were the control variable, did not find a significant difference between the experimental and control groups. Therefore, participants in the experimental group had lower pre-test scores (M=1.430) than students in the control group (M=1,526), but this difference was not significant (F(1.176)=6.32, p=.50).

Table 7.

ANCOVA results for "Basic Functions"

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	38.211a	2	19.10	13.77	.00	.13
Intercept	53.394	1	53.39	38.49	.00	.18
Pre-Test	38.211	1	38.21	27.55	.00	.14
Group	.002	1	.00	.00	.97	.00
Error	239.970	173	1.39			
Total	860.000	176				
Corrected Total	278.182	175				

Table 7 summarizes the results of ANCOVA in terms of "Basic Functions". According to the ANCOVA analysis, in which the post-test scores of the Basic Functions were the dependent variable, and the pre-test results were the control variable, no significant difference was found between the experimental and control groups. According to this table, although the participants in the experimental group (M=1.815) scored lower than the students in the control group (M=1.821), this difference was not significant (F(1.176)=13.77, p=.97).

4.2. Computational Thinking Self-Efficacy

Table 8.

ANCOVA results in terms of Algorithm Design Self-Efficacy.

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	729.757a	2	364.88	18.59	.00	.18
Intercept	2049.913	1	2049.91	104.42	.00	.38
Pre-Test	687.004	1	687.00	34.99	.00	.17
Group	13.823	1	13.82	.70	.40	.00
Error	3396.237	173	19.63			
Total	68539.000	176				
Corrected Total	4125.994	175				

Table 8 summarizes the results of ANCOVA in terms of Algorithm Design Self-Efficacy. According to the ANCOVA analysis, in which the post-test scores of Algorithmic Design Self-Efficacy were the dependent variable, and the pre-test results were the control variable, no significant difference was found between the experimental and control groups. Despite scoring higher than the students in the control group (M=18.846), the participants in the experimental group (M=19.409) did not have a significant difference (F(1.176)=18.59, p=.40).

Table 9.

ANCOVA results in terms of Problem-Solving Efficacy

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	573.810a	2	286.90	29.01	.00	.25
Intercept	334.666	1	334.67	33.84	.00	.16
Pre-Test	560.393	1	560.39	56.67	.00	.25
Group	17.996	1	18.00	1.82	.18	.01
Error	1710.826	173	9.89			
Total	106172.000	176				
Corrected Total	2284.636	175				

Table 9 summarizes the results of ANCOVA in terms of Problem-Solving Self Efficacy. According to the ANCOVA analysis, in which the post-test scores of Problem-Solving Competence were the dependent variable, and the pre-test results were the control variable, no significant difference was found between the experimental and control groups. When controlling for pre-test results, the participants in the experimental group (M=23.979) scored lower than the students in the control group (M=24.619), but this difference was not significant (F(1.176)=29.01, p=.18).

Table 10.

ANCOVA results in terms of the Data Processing Efficacy

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	260.643a	2	130.32	11.95	.00	.12
Intercept	1611.190	1	1611.19	147.75	.00	.46
Pre-Test	247.865	1	247.86	22.73	.00	.12
Group	12.545	1	12.54	1.15	.28	.01
Error	1886.578	173	10.90			
Total	46027.000	176				
Corrected Total	2147.222	175				

Table 10 summarizes the results of the ANCOVA in terms of the dimension of Data Processing Efficacy. In the ANCOVA, the Data Processing efficacy final test scores constituted the dependent variable, and the pre-test results were the control variable. No significant difference was found between the experimental and control groups. Participants in the experimental group (M=16.054) scored higher on the pre-test than the control group (M=15.520), but this difference was not significant at a meaningful level (F(1.176)=11.95, p=.28).

Table 11.

ANCOVA results in terms of the Basic Programming Efficacy

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	171.025a	2	85.51	14.05	.00	.14
Intercept	817.348	1	817.35	134.28	.00	.44
Pre-Test	156.330	1	156.33	25.68	.00	.13
Group	11.012	1	11.01	1.81	.18	.01
Error	1053.015	173	6.09			
Total	22807.000	176				
Corrected Total	1224.040	175				

Table 11 summarizes the results of the ANCOVA in terms of the dimension of the Basic Programming Efficacy. The ANCOVA results showed no significant difference between the experimental and control groups; the Basic Programming Competence final test scores were the dependent variable, and the pre-test results were the control variable. According to this table, although participants in the experimental group (M=11.321) scored higher on the pre-test than the students in the control group (M=10.821), this difference was not significant at a meaningful level (F(1.176)=14.05, p=.18).

Table 12.

ANCOVA results in terms of Self-Confidence

Source	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared
Corrected Model	99.487a	2	49.74	11.51	.00	.12
Intercept	487.262	1	487.26	112.70	.00	.39
Pre-Test	92.962	1	92.96	21.50	.00	.11
Grup	4.732	1	4.73	1.09	.30	.01
Error	747.945	173	4.32			
Total	25952.000	176				
Corrected Total	847.432	175				

Table 12 summarizes the results of the ANCOVA in terms of the dimension of the Self-Confidence ANCOVA in terms of the dimension of the Self-Confidence. No significant difference was found between the experimental and control groups with control in the pre-test scores. Participants in the experimental group (M=11.781) scored lower on the pre-test when the results of the control group (M=12.109) were controlled, but this difference was not significant at (F(1.176)=11.51, p=.30).

5. Discussion and Conclusion

Studies have been conducted to understand the role of information technology and mathematics lessons in acquiring computational thinking skills (Cui & Ng, 2021; Ng & Cui, 2021; Sung & Black, 2020). This study examines the role of activities that present the gains from these two subjects together in developing Computational Thinking skills. Toward this end, activities designed for middle school 6th-grade students were implemented. Experimental group students applied supported activities on topics included in the teaching program of the first term mathematics lesson order of operations, negative numbers, division rules, and prime numbers) using the Scratch program, while the control group students carried out game design activities according to the information technology and software teaching program using the Scratch program. In the research in which the pre-test-post-test-control-group design was used, the students' mean scores obtained from CTT and CTSES were compared.

5.1. Computational Thinking Skills

In the CTT scope, there are seven sub-dimensions: basic sorting, loops that repeat a specific number of times, loops that run until a condition is met, simple conditional statements and complex conditional statements, loops that run as long as the condition is true and simple

functions. The experimental process result indicated no statistically significant difference between the experimental and control group students in any dimension. Since the basic sorting dimension consists of relatively easy questions that include coding (move forward, turn right, and turn left) operations at an introductory level, it can be thought that both groups scored at similar levels. In addition, existing pedagogical and methodological might not provide a fundamental to merge math and introductory computer science education (Nordby et al., 2022).

Students in both groups have carried out block-based coding activities in their 5th-grade Information Technology and Software classes and are experienced in this regard. Indeed, Çetin et al. (2020) showed that as the class level increases, Computational Thinking scores also increase accordingly. Therefore, taking information technology-oriented lessons and having content that can contribute to computational thinking in different lessons may have also positively affected students' natural cognitive development process. As in many points reached in the study, this situation may have caused no significant difference between the experimental and control groups in the basic sorting dimension. According to Durak and Saritepeci (2018, p. 200), computational thinking is highly predicted by ways of thinking, maths class academic success, attitude against maths class, level of education, science class academic success, information technologies academic success, attitude against information technologies class, sex, IT usage experience, period of daily internet use and attitude against science class. This situation may have also reduced the effect of the experimental process.

When the pre-test scores were controlled, there was no significant difference between the experimental and control group students in the section on loops that repeated a specific number of times. However, the students in the experimental group scored higher, although not at a significant level, than those in the control group. This dimension consists of questions that require a specific operation or operations to be repeated a specific number of times. These questions require the student to identify repeating patterns, determine the number of repetitions, and order the codes accordingly if any errors exist. Therefore, it can be thought that the activities with mathematics support in the experimental group positively affected the student's performance in this field. As in the basic sorting dimension, when the student's natural development is thought to affect the computational thinking skill, the possibility of obtaining a non-significant but higher difference in the loop dimension in a purified environment from these effects emerges. Therefore, using concepts such as pattern recognition

in computer science can contribute significantly to information processing thinking skills. In fact, within the scope of CT skills, students must recognize when algorithm steps are repeated, while at the same time, it is common in mathematics to repeat a primary step, such as adding a unit to achieve a broader goal or placing a length unit in order to perform a task. The relationship between these two disciplines is thought to be synergistic (Rich et al., 2019).

The dimension of "loops with a specific number of repetitions" consists of repetitive operations that require proper ordering until a condition is met. When controlling for pre-test scores in this dimension, it was found that although the experimental group of students scored higher than the control group, the difference was insignificant. For example, in an activity related to prime numbers, the students in the experimental group used the concept of loops to determine whether a number is prime or not while adhering to certain conditions. In this context, it can be argued that such activities support students' development in this area. Indeed, it can be said that activities in the context of mathematics classes have the potential to be transferred to programming education in terms of being concrete, containing concepts encountered in daily life, and being frequently used, for example, when subtracting a large number from a small number, there is a back-repeating counting process and a stopping point (Cui & Ng, 2021).

No significant difference between the experimental and control groups regarding simple and complex conditional statements was found. However, the students in the control group did score higher than the experimental group, albeit not significantly. In that dimension, simple conditional statements were asked but presented within repetitive loops until the condition was met. Similarly, in the questions of the complex conditional statements dimension, the questions were given again within repetitive loops until the condition was met. However, the condition structure was presented with multiple options: "If...then... Else...". It is possible that the activities the students in the control group applied had more conditional statements, which may have led to more support for these students in this area. Indeed, Cui and Ng (2021) pointed out that students' difficulties in learning a computational thinking environment and in the process that includes mathematical concepts and problem-solving applications are combined. Therefore, in mathematics-supported programming education, the experimental group students' effort to learn mathematical and programming concepts together may lead them to choose various ways in terms of prioritizing mathematics or programming. In this context, experimental group students might give priority to mathematical concepts. In addition, the

limited presence of essential mathematical topics and topics containing simple or complex conditional statements in the education process that students have received until 6th grade may have limited the potential difference to be in favor of the experimental group. Therefore, conditional statements in the created mathematical activities may have been limited. As mentioned above, the presence of game design-based activities in the control group and the need for many conditional structures from simple to complex in the nature of games may have led to the control group scoring higher, even if the result was not significant.

5.2. Computational Thinking Self-Efficacy

The Algorithm Design self-efficacy includes topics such as what an algorithm is, creating simple and conditional algorithms, predicting the algorithm's output, and debugging. It was observed that the views of the students in the experimental group on these topics were higher than those of the control group, even if the difference was not statistically significant. While solving mathematical problems, students naturally perform algorithm design stages, which may have led the experimental group students to see themselves as more competent in algorithms. Indeed, Lockwood et al. (2016) defined algorithmic thinking as a logical, organized way of breaking down a complex goal into a series of (sequential) steps using existing tools. It can be argued that mathematics-supported programming activities contain more concrete examples that support students' algorithmic thinking skills.

When examining the pre-test results in the dimension of Problem-Solving self-efficacy, it is observed that the students in the experimental group scored lower than the control group students, but the difference is insignificant. This dimension includes topics related to problem-solving skills. Mathematics is one of the most challenging subjects for students. The perception of mathematics as brutal may also have led to a decreased perception of problem-solving skills. The cognitive load of the primary programming education taught with mathematical activities may have increased for students, reducing their perception of problem-solving competency. Similarly, Psycharis and Kallia (2017) found that mathematics and programming education did not significantly affect students' problem-solving skills. Therefore, more research is needed to clarify the reasons for this situation.

The questions in the Basic Programming self-efficacy include topics such as variables, conditional structures, loops, and arithmetic operators. The students in the experimental group

scored higher, albeit not at a statistically significant level, than the control group students. This could be attributed to the fact that the mathematical activities included in the experimental group also involved the use of arithmetic operators, which may have increased the students' confidence in their abilities related to basic programming skills. Opposite results were gained from the examination of the self-confidence dimension. The control group students' firmer belief in their programming abilities may be attributed to their exposure to more complex activities and more remarkable development in abstraction, decomposition, algorithmic thinking, and problem-solving while designing games.

Despite the insignificant differences and reasons discussed in the self-efficacy, it should be noted that these differences are pretty slight and that a variety of factors, such as the environment in which the activity took place, different variables related to the students, and information learned in other classes may have contributed to the slight differences observed. Additionally, the differences that emerged may have been random and have the potential to evolve differently in repeated measurements. Therefore, it is recommended to approach the relevant results in the aforementioned situations with caution. Due to the commonalities in the nature of mathematics and programming, it is recommended that in-depth studies should be carried out. In this context, the relationship between mathematics and each dimension of computational thinking skills should be focused. In particular, the experimental investigation of cognitive skills such as algorithmic thinking and problem-solving and the investigation of the effect of programming activities will reveal essential findings in the field.

6. References

Adsay, C., Korkmaz, Ö., Çakır, R., & Erdoğmuş Uğur, F. (2020). Ortaokul öğrencilerinin blok temelli kodlama eğitimine dönük öz-yeterlik algı düzeyleri, STEM ve bilgisayarca düşünme beceri düzeyleri. *Eğitim Teknolojisi Kuram ve Uygulama, 10*(2), 469-489.

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

Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., & Zagani, J. (2016). A K-6 Computational Thinking Curriculum Framework: *Implication for Teacher Knowledge. Educational Technology & Society, 19*(3), 47-57. <Go to ISI>://WOS:000383353700006

Ataman-Uslu, N., Mumcu, F., & Eğin, F. (2018). The effect of visual programming activities on secondary school students' computational thinking skills. *Journal of Ege Education Technologies, 2*(1), 19-31.

Cui, Z. H., & Ng, O. L. (2021). The Interplay Between Mathematical and Computational Thinking in Primary School Students' Mathematical Problem-Solving Within a Programming Environment. *Journal of Educational Computing Research, 59*(5), 988–1012. https://doi.org/Artn 0735633120979930 10.1177/0735633120979930

Curzon, P. (2015). *Computational thinking: Searching to speak*. Queen Mary, University of London.

Çetin, I., Otu, T., & Oktaç, A. (2020). Adaption of the computational thinking test into Turkish. *Turkish Journal of Computer and Mathematics Education (TURCOMAT), 11*(2), 343-360.

Çubukluöz, Ö. (2019). 6. sınıf öğrencilerinin matematik dersindeki öğrenme zorluklarının Scratch programıyla tasarlanan matematiksel oyunlarla giderilmesi: bir eylem araştırması Bartın Üniversitesi, Eğitim Bilimleri Enstitüsü].

Dolmacı, A., & Akhan, N. E. (2020). Bilişimsel Düşünme Becerileri Ölçeğinin Geliştirilmesi: Geçerlik ve Güvenirlik Çalışması. *Itobiad: Journal of the Human & Social Science Researches, 9*(3).

Durak, H. Y., & Saritepeci, M. (2018). Analysis of the relation between computational thinking skills and various variables with the structural equation model. *Computers & Education*, 116, 191-202.

Fraenkel, J. R., Wallen, N. E., & Hyun, H. H. (2012). *How to Design and Evaluate Research in Education.* McGraw-Hill.

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/10.3102/0013189x12463051

Gulbahar, Y., & Kalelioglu, F. (2015). Competencies for e-Instructors: How to qualify and guarantee sustainability. *Contemporary Educational Technology, 6*(2), 140-154.

Gülbahar, Y., & Kalelioğlu, F. (2018). Bilişim teknolojileri ve bilgisayar bilimi: Öğretim programı güncelleme süreci. *Millî Eğitim Dergisi, 47*(217), 5-23.

Gülbahar, Y., Kert, S. B., & Kalelioğlu, F. (2019). Bilgi işlemsel düşünme becerisine yönelik öz yeterlik algısı ölçeği: Geçerlik ve güvenirlik çalışması. *Turkish Journal of Computer and Mathematics Education (TURCOMAT), 10*(1), 1-29.

Hsu, T. C., Chang, S. C., & Hung, Y. T. (2018). How to learn and how to teach computational thinking: Suggestions based on a review of the literature. *Computers & Education, 126*, 296–310. https://doi.org/10.1016/j.compedu.2018.07.004

ISTE. (2016). 2016 ISTE Standards for Students.

Korkmaz, Ö., Çakır, R., & Özden, M. Y. (2015). Bilgisayarca düşünme beceri düzeyleri ölçeğinin (bdbd) ortaokul düzeyine uyarlanması. *Gazi Eğitim Bilimleri Dergisi, 1*(2), 143-162.

Kukul, V., & Karatas, S. (2019). Computational Thinking Self-Efficacy Scale: Development, Validity and Reliability. *Informatics in Education, 18*(1), 151-164. https://doi.org/10.15388/infedu.2019.07

Lockwood, E., DeJarnette, A. F., Asay, A., & Thomas, M. (2016). Algorithmic Thinking: An Initial Characterization of Computational Thinking in Mathematics. North American Chapter of the International Group for the Psychology of Mathematics Education.

Lockwood, J., & Mooney, A. (2017). Computational thinking in education: Where does it fit? A systematic literary review. arXiv preprint arXiv:1703.07659.

Ng, O.-L., & Cui, Z. (2021). Examining primary students' mathematical problem-solving in a programming context: Towards computationally enhanced mathematics education. *ZDM–Mathematics Education, 53*(4), 847-860.

Nordby, S. K., Bjerke, A. H., & Mifsud, L. (2022). Computational thinking in the primary mathematics classroom: A systematic review. *Digital Experiences in Mathematics Education, 8*(1), 27-49.

Okuducu, A. (2020). Scratch destekli matematik öğretiminin 6. sınıf öğrencilerinin cebirsel ifadeler konusundaki akademik başarılarına ve tutumlarına etkisi Fen Bilimleri Enstitüsü].

Oluk, A., Korkmaz, Ö., & Oluk, H. A. (2018). Scratch'ın 5. sınıf öğrencilerinin algoritma geliştirme ve bilgi-işlemsel düşünme becerilerine etkisi. *Turkish Journal of Computer and Mathematics Education (TURCOMAT), 9*(1), 54-71.

Özçınar, H. (2017). Hesaplamali düşünme araştirmalarinin bibliyometrik analizi. *Eğitim Teknolojisi Kuram ve Uygulama, 7*(2), 149-171.

Özmen, B. (2016). Ortaokul öğrencilerine yönelik bilgi işlemsel düşünme becerileri testinin geliştirilmesi: Geçerlik ve güvenirlik çalışması. 4th International Instructional Technologies & Teacher Education Symposium.

Papert, S. (1980). *Children, computers, and powerful ideas*. Harvester Press (United Kingdom). 978–973.

Papert, S. (1996). An exploration in the space of mathematics education. *Int. J. Comput. Math. Learn., 1*(1), 95–123.

Psycharis, S., & Kallia, M. (2017). The effects of computer programming on high school students' reasoning skills and mathematical self-efficacy and problem-solving. *Instructional Science, 45*(5), 583-602. https://doi.org/10.1007/s11251-017-9421-5

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for All. *Communications of the ACM, 52*(11), 60-67. https://doi.org/10.1145/1592761.1592779

Rich, K. M., Spaepen, E., Strickland, C., & Moran, C. (2019). Synergies and differences in mathematical and computational thinking: implications for integrated instruction. *Interactive Learning Environments, 28*(3), 272–283. https://doi.org/10.1080/10494820.2019.1612445

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.

Román-González, M., Pérez-González, J.-C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in human behavior, 72*, 678-691.

Sung, W., Ahn, J., & Black, J. B. (2017). Introducing Computational Thinking to Young Learners: Practicing Computational Perspectives Through Embodiment in Mathematics Education. *Technology Knowledge and Learning, 22*(3), 443-463. https://doi.org/10.1007/s10758-017-9328-x

Sung, W., & Black, J. B. (2020). Factors to consider when designing effective learning: Infusing computational thinking in mathematics to support thinking-doing. *Journal of Research on Technology in Education, 53*(4), 404–426. https://doi.org/10.1080/15391523.2020.1784066

Syslo, M. M., & Kwiatkowska, A. B. (2013). Informatics for All High School Students-A Computational Thinking Approach. In *International conference on informatics in schools: Situation, evolution, and perspectives* (pp. 43-56).

Şahiner, A., & Kert, S. B. (2016). Komputasyonel düşünme kavramı ile ilgili 2006-2015 yılları arasındaki çalışmaların incelenmesi. *Avrupa Bilim ve Teknoloji Dergisi, 5*(9), 38-43.

Şimşek, E. (2018). Programlama öğretiminde robotik ve scratch uygulamalarının öğrencilerin bilgi işlemsel düşünme becerileri ve akademik başarılarına etkisi. Yayımlanmamış yüksek lisans tezi.

Tosik Gün, E., & Güyer, T. (2019). Bilgi işlemsel düşünme becerisinin değerlendirilmesine ilişkin sistematik alanyazın taraması. *Ahmet Keleşoğlu Eğitim Fakültesi Dergisi, 1*(2), 99-120. https://doi.org/https://doi.org/10.38151/akef.597505

Vatansever, Ö. (2018). Scratch ile Programlama Öğretiminin Ortaokul 5. ve 6. Sınıf Öğrencilerinin Problem Çözme Becerileri Üzerindeki Etkisinin Incelenmesi (Doctoral dissertation, Bursa Uludag University (Turkey).

Weinberg, A. E. (2013). Computational thinking: An investigation of the existing scholarship and research Colorado State University].

Wilkerson, M. H., & Fenwick, M. (2017). Using mathematics and computational thinking. Helping students make sense of the world using next generation science and engineering practices, 181-204.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35. https://doi.org/Doi 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.

Yünkül, E., Durak, G., Çankaya, S., & Abidin, Z. (2017). The effects of scratch software on students' computational thinking skills. *Necatibey Faculty of Education Electronic Journal of Science and Mathematics Education, 11*(2), 502-517.