

THE INVESTIGATION OF PSYCHO-EDUCATIONAL CONSTRUCTS IN
RELATION TO MIDDLE SCHOOL STUDENTS' LEARNING OF BASIC
COMPUTER PROGRAMMING CONCEPTS

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

BY

PINAR KEFELİ BERBER

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

JULY 2024

Approval of the thesis:

**THE INVESTIGATION OF PSYCHO-EDUCATIONAL CONSTRUCTS IN
RELATION TO MIDDLE SCHOOL STUDENTS' LEARNING OF BASIC
COMPUTER PROGRAMMING CONCEPTS**

submitted by **PINAR KEFELİ BERBER** 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. Naci Emre Altun
Dean, **Graduate School of Natural and Applied Sciences**

Prof. Dr. Soner Yıldırım
Head of the Department, **Comp. Edu. and Inst. Tech.**

Prof. Dr. Soner Yıldırım
Supervisor, **Comp. Edu. and Inst. Tech., METU**

Examining Committee Members:

Prof. Dr. Halil Yurdugül
Comp. Edu. and Inst. Tech., Hacettepe University

Prof. Dr. Soner Yıldırım
Comp. Edu. and Inst. Tech., METU

Prof. Dr. Ömer Delialioğlu
Comp. Edu. and Inst. Tech., METU

Assoc. Prof. Dr. Evren Şumuer
Comp. Edu. and Inst. Tech., Kocaeli University

Assoc. Prof. Dr. Erkan Er
Comp. Edu. and Inst. Tech., METU

Date: 05.07.2024



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 : Pınar Kefeli Berber

Signature :

ABSTRACT

THE INVESTIGATION OF PSYCHO-EDUCATIONAL CONSTRUCTS IN RELATION TO MIDDLE SCHOOL STUDENTS' LEARNING OF BASIC COMPUTER PROGRAMMING CONCEPTS

Kefeli Berber, Pınar

Doctor of Philosophy, Computer Education and Instructional Technology

Supervisor: Prof. Dr. Soner Yıldırım

July 2024, 296 pages

The main purpose of this study was to investigate the cognitive and motivational factors contributing to the acquisition of fundamental computer programming concepts at the middle school level. Employing a mixed-method embedded design approach, the study aimed to explore how personal achievement goal orientations, perceived classroom goal structures, academic-related perceptions, beliefs and strategies, attitude towards coding education, cognitive load, mathematics and reading comprehension achievement, gender, and geographical school location predict students' achievement scores in programming. Participants of the study included 199 fifth-grade students. The implementation was conducted over ten weeks in the Information Technologies and Software course across three public middle schools. Data were collected through surveys, achievement tests, and interviews. Data analysis involved both quantitative and qualitative approaches. For the analysis of quantitative data, various methods were employed, including independent samples t-test, repeated measures ANOVA, mulrank function, doubly repeated MANOVA, and hierarchical regression. The study's results revealed that

mathematics achievement was the strongest predictor of programming achievement, followed by reading comprehension achievement, geographic school location, extraneous load, attitude towards coding education, and academic self-handicapping strategies. Furthermore, changes were observed in students' cognitive load levels throughout the programming learning process, particularly concerning certain programming topics. While the results indicated no significant differences based on gender regarding the variables investigated, significant differences were found concerning geographical school location in terms of academic achievement and motivational factors. Students' experiences with programming and the online coding platform used in the research were generally positive.

Keywords: Computer Programming Education for Children, Coding, Cognitive Load, Motivation

ÖZ

ORTAOKUL ÖĞRENCİLERİNİN TEMEL BİLGİSAYAR PROGRAMLAMA KAVRAMLARINI ÖĞRENMELERİNE İLİŞKİN PSİKO-EĞİTSEL YAPILARIN İNCELENMESİ

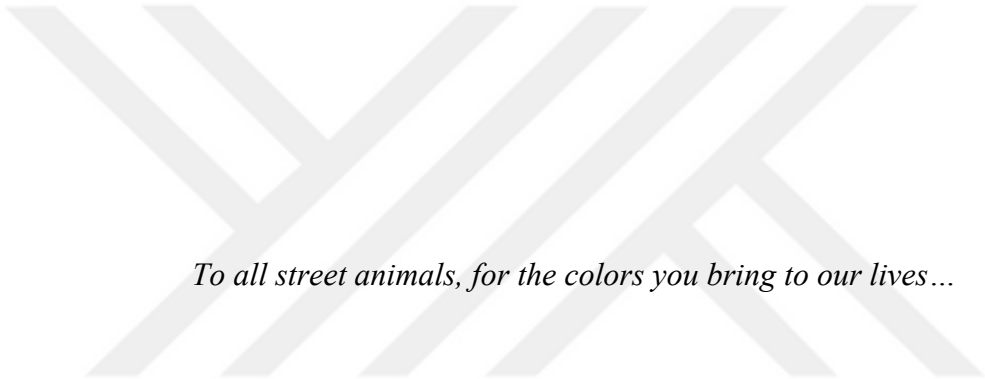
Kefeli Berber, Pınar
Doktora, Bilgisayar ve Öğretim Teknolojileri Eğitimi
Tez Yöneticisi: Prof. Dr. Soner Yıldırım

Temmuz 2024, 296 sayfa

Bu çalışmanın temel amacı, ortaokul düzeyinde temel bilgisayar programlama kavramlarının kazanımına katkıda bulunan bilişsel ve motivasyonel faktörleri araştırmaktır. Karma yöntemli gömülü tasarım yaklaşımının kullanıldığı bu çalışmada, kişisel başarı hedef yönelimleri, algılanan sınıf hedef yapıları, akademik algılar, inançlar ve stratejiler, kodlama eğitimine yönelik tutum, bilişsel yük, matematik ve okuduğunu anlama başarısı, cinsiyet ve coğrafi okul konumunun öğrencilerin programlama başarı puanlarını nasıl etkilediği incelenmiştir. Çalışmanın katılımcılarını 199 beşinci sınıf öğrencisi oluşturmaktadır. Uygulama, üç devlet ortaokulunda, on hafta boyunca Bilişim Teknolojileri ve Yazılım dersi kapsamında gerçekleştirilmiştir. Veriler anketler, başarı testleri ve mülakatlar yoluyla toplanmıştır. Verilerin analizinde hem nicel hem de nitel yaklaşımlardan faydalanılmıştır. Nicel verilerin analizi için bağımsız örneklem t-testi, tekrarlı ölçümler ANOVA, mulrank fonksiyonu, iki yönlü tekrarlı ölçümler MANOVA ve hiyerarşik regresyon gibi çeşitli yöntemler kullanılmıştır. Çalışmanın sonuçları, matematik başarısının programlama başarısının en güçlü yordayıcısı olduğunu, bunu okuduğunu anlama başarısı, coğrafi okul konumu, konu dışı yük, kodlama eğitime

yönelik tutum ve akademik kendini engelleme stratejilerinin takip ettiğini ortaya koymuştur. Ayrıca, programlama öğrenme sürecinde, özellikle belirli programlama konularıyla ilgili olarak öğrencilerin bilişsel yük seviyelerinde değişiklikler gözlemlenmiştir. Sonuçlar, incelenen değişkenler açısından cinsiyete göre anlamlı bir fark olmadığını gösterirken, coğrafi okul konumuna göre akademik başarı ve motivasyonel faktörler açısından anlamlı farklılıklar elde edilmiştir. Öğrencilerin programlama ve araştırmada kullanılan çevrimiçi kodlama platformuna yönelik deneyimlerinin genellikle olumlu olduğu görülmüştür.

Anahtar Kelimeler: Çocuklar için Bilgisayar Programlama Eğitimi, Kodlama, Bilişsel Yük, Motivasyon



To all street animals, for the colors you bring to our lives...

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to my supervisor, Prof. Dr. Soner Yıldırım, for his invaluable patience, guidance, and feedback throughout this research. This endeavor would not have been possible without his support.

I also could not have undertaken this journey without the knowledge and expertise of my dissertation committee members, Prof. Dr. Ömer Delialioğlu and Assoc. Prof. Dr. Evren Şumuer. Additionally, I extend my heartfelt thanks to my jury members, Prof. Dr. Halil Yurdugül and Assoc. Prof. Dr. Erkan Er, for their insightful contributions.

Words cannot express my gratitude to Assoc. Prof. Dr. Sacip Toker who generously provided knowledge and expertise during data analysis process.

I would like to express my appreciation to the school administrators, teachers, and students who participated in this study. Their collaboration and support were essential to the research process. I am particularly grateful to Vesile Çelik for her assistance during data collection. I am deeply indebted to Kemal Gündüz for his unwavering support throughout the pilot and main studies. I had the pleasure of working with each of them.

I would like to express my sincere gratitude to my colleague, Nuray Toplu, for her exceptional work as a second coder in the qualitative analysis. Her dedication and support were invaluable. I would also like to thank Aslı Boyraz for her timely assistance. I am grateful for the support and encouragement of my friends and colleagues, who were always there to offer guidance and support.

I would like to express my heartfelt gratitude to my family for their unconditional love and support. My mother, Belgin Kefeli, has always believed in me; my father, İlkey Kefeli, has been a constant source of support; my grandparents, Abbas Kefeli

and Perihan Kefeli, have shaped me with their love; and my sister, Bahar Kefeli Çol, has always been by my side. My husband, Kasım Berber, has been an incredible source of strength and encouragement throughout this journey, and I am deeply thankful for his unwavering support and understanding.

Lastly, I would like to thank my beloved children, Lokum and Piços, who brought joy and inspiration to my life during this challenging journey.



TABLE OF CONTENTS

ABSTRACT	v
ÖZ.....	vii
ACKNOWLEDGMENTS.....	x
TABLE OF CONTENTS	xii
LIST OF TABLES	xvii
LIST OF FIGURES.....	xix
LIST OF ABBREVIATIONS	xx
CHAPTERS	
1 INTRODUCTION.....	1
1.1 Background of the Study	1
1.2 Purpose of the Study.....	8
1.3 Research Questions	9
1.4 Significance of the Study.....	11
2 LITERATURE REVIEW	15
2.1 Cognitive Load Theory.....	15
2.1.1 Human Cognitive Architecture.....	15
2.1.2 Foundations of Cognitive Load Theory	21
2.1.3 Types of Cognitive Load	23
2.1.4 Research on the Role of Cognitive Load Theory in Computer Programming Education.....	27
2.2 Motivation and Learning	30
2.2.1 Goal Orientation Theory.....	30

2.2.2	Self-efficacy	33
2.2.3	Academic Self-Handicapping Strategies	34
2.2.4	Attitude	36
2.2.5	Cheating Behavior	37
2.2.6	Research on the Impact of Motivational Factors on Students' Learning of Computer Programming	38
2.3	Programming Education for Young Learners	40
2.3.1	Block-based Programming Environments	43
2.4	Summary	45
3	METHODOLOGY	49
3.1	Research Questions	49
3.2	Participants	51
3.2.1	Participants in the Quantitative Phase	51
3.2.2	Participants in the Qualitative Phase	54
3.3	Research Design of the Study	55
3.4	Procedure of the Study	58
3.4.1	Preliminary Investigation	58
3.4.2	Adaption Process of the Lesson Plans	62
3.4.3	Lesson Plan Evaluation Workshop	63
3.5	Data Collection Instruments	64
3.5.1	Coding Achievement Test	64
3.5.2	Cognitive Load Scale	71
3.5.3	Patterns of Adaptive Learning Scales (PALS)	79
3.5.4	Attitudes Towards Coding Education Scale (ATCES)	79

3.5.5	Reading Comprehension Achievement Test	80
3.5.6	5th Grade Mathematics Achievement Test	80
3.5.7	Student Interview Protocol	80
3.6	Pilot Study	82
3.7	Implementation of the Study	85
3.8	Data Analysis.....	86
3.8.1	Quantitative Data Analysis.....	86
3.8.2	Qualitative Data Analysis.....	87
3.9	Trustworthiness for Qualitative Part of the Study	89
3.9.1	Internal Validity (Credibility).....	89
3.9.2	External Validity (Transferability)	91
3.9.3	Researcher Role and Bias	92
3.10	Ethical Issues	93
3.11	Limitations of the Study	94
4	RESULTS.....	97
4.1	Results of the Quantitative Data Analysis.....	97
4.1.1	Correlation Between Variables of the Study	97
4.1.2	Results of the Research Question 1	100
4.1.3	Results of the Research Question 2	103
4.1.4	Results of the Research Question 3	108
4.1.5	Results of the Research Question 4	114
4.2	Results of the Qualitative Data Analysis.....	122
4.2.1	Results of the Research Question 5	122
5	DISCUSSION AND CONCLUSION	187

5.1	Major Findings and Discussion	187
5.1.1	Cognitive Load.....	187
5.1.2	Gender.....	192
5.1.3	Geographical School Location.....	195
5.1.4	Mathematics Skills.....	197
5.1.5	Reading Comprehension Skills.....	201
5.1.6	Attitude Toward Programming.....	204
5.1.7	Patterns of Adaptive Learning	207
5.2	Conclusion	208
5.3	Implications of the Findings	212
5.4	Recommendations for Further Research.....	216
REFERENCES		217
APPENDICES		249
A.	Lesson Plan Evaluation Form	249
B.	Coding Achievement Test.....	251
C.	Cognitive Load Scale	257
D.	Patterns of Adaptive Learning Scale.....	258
E.	Attitudes Toward Coding Education Scale.....	261
F.	Reading Comprehension Achievement Test.....	263
G.	5th Grade Mathematics Achievement Test.....	269
H.	Interview Protocol.....	271
I.	Sample Lesson Plan	273
J.	Approval of Human Subjects Ethics Committee at METU - I	275
K.	Approval of Human Subjects Ethics Committee at METU - II.....	276

L. Approval of Provincial Directorate of National Education.....	277
M. The Original Turkish Versions of the Quotes	278
CURRICULUM VITAE	295



LIST OF TABLES

TABLES

Table 3.1 Participants of the Study by Schools	51
Table 3.2 Characteristics of the Participants.....	52
Table 3.3 Parental Education Level of the Participants.....	53
Table 3.4 Demographic and School Information of Interviewed Students	55
Table 3.5 Content Validity Values of the Test Items.....	66
Table 3.6 Item Analysis Results of the Coding Achievement Test	69
Table 3.7 The Distribution of the Items According to Learning Objectives	70
Table 3.8 Items of the Cognitive Load Scale.....	71
Table 3.9 Distribution of Participants to Schools	73
Table 3.10 Normality Distribution of The Cognitive Load Scale Scores.....	74
Table 3.11 Descriptive Statistics.....	75
Table 3.12 Descriptive Statistics for Subfactors.....	75
Table 3.13 Factor Correlations	77
Table 3.14 Factor Loadings of the Items	77
Table 3.15 Reliability Analysis Results of the CLS	79
Table 3.16 Weekly Learning Objectives, Lesson Plans and Activities	83
Table 3.17 Data Collection Procedures	86
Table 4.1 Correlation Coefficients Between the Variables.....	99
Table 4.2 Descriptive Statistics for Repeated Cognitive Load Measures Across Seven Programming Concepts	101
Table 4.3 Results of One-way Repeated Measures ANOVA Comparing Cognitive Load Scores Across Seven Different Programming Concepts	102
Table 4.4 Results of t-test and Descriptive Statistics for ATCE, MA, RCA and CA by Gender.....	104
Table 4.5 Descriptive Statistics for Cognitive Load Measures for Gender Across Seven Programming Concepts	106

Table 4.6 Results of Doubly Repeated MANOVA for Cognitive Load Types by Gender	107
Table 4.7 Mann-Whitney U Test Results for PALS by Geographical School Location.....	109
Table 4.8 Results of t-test and Descriptive Statistics for ATCE, MA, RCA and CA by Geographical School Location	110
Table 4.9 Descriptive Statistics for Cognitive Load Measures for School Location Across Seven Programming Concepts	111
Table 4.10 Results of Doubly Repeated MANOVA for Cognitive Load Types by Geographical School Location	112
Table 4.11 Collinearity Statistics of the Predictor Variables	115
Table 4.12 Standardized Residual Statistics.....	116
Table 4.13 Four-Step Hierarchical Multiple Regression Analysis Results	120
Table 4.14 Distribution of Code Frequencies by the Theme of Cognitive Demands and Instructional Factors	124
Table 4.15 Distribution of Code Frequencies by the Theme of Effective Instructional Approaches.....	137
Table 4.16 Distribution of Code Frequencies by the Theme of Collaborative Learning Approaches.....	147
Table 4.17 Distribution of Code Frequencies by the Theme of Independent Learning Approaches.....	162
Table 4.18 Distribution of Code Frequencies by the Theme of Goal Setting	168
Table 4.19 Distribution of Code Frequencies by the Theme of Affective Aspects	177

LIST OF FIGURES

FIGURES

Figure 3.1. Scree Plot for CLS	76
Figure 3.2. Path Analysis Diagram for CLS within CFA	78
Figure 3.3. First Cycle and Second Cycle Coding Methods.....	88
Figure 4.1 Plot of Estimated Marginal Means of Intrinsic Load by Gender	113
Figure 4.2 Normal Probability Plot (P- P) of the Regression Standardized Residual for CA	117
Figure 4.3 Histogram of Regression Standardized Residual for CA	117
Figure 4.4 Scatterplot of Standardized Residuals and Standardized Predicted Values	118
Figure 4.5 Themes and Their Corresponding Categories	123
Figure 4.6 (a) Sample programming task on code.org (Course F-Lesson 8: Nested Loops in Maze/Level 10) about nested loops and (b) possible solution to this task	128
Figure 4.7 (a) Sample programming task on code.org (Course 2 – Lesson 13: Bee Conditionals/Level 8) about conditional statements and (b) its solution.....	133

LIST OF ABBREVIATIONS

ABBREVIATIONS

CLS: Cognitive Load Scale

IL: Intrinsic Load

EL: Extraneous Load

GL: Germane Load

ATCES: Attitudes Toward Coding Education Scale

PALS: Patterns of Adaptive Learning Scale

MA: Math Achievement

RCA: Reading Comprehension Achievement

CA: Coding Achievement

MGO: Mastery Goal Orientation

PAPGO: Performance-Approach Goal Orientation

PAVGO: Performance-Avoid Goal Orientation

AE: Academic Efficacy

ASHS: Academic Self Handicapping Strategies

CB: Cheating Behavior

CMGS: Classroom Mastery Goal Structure

CPAPGS: Classroom Performance-Approach Goal Structure

CPAVGS: Classroom Performance-Avoid Goal Structure

CA: Coding Achievement

CHAPTER 1

INTRODUCTION

This study aims to investigate the factors influencing the academic achievement of middle school students in learning fundamental computer programming concepts. The introduction provides a contextual background, outlines the study's purpose and research questions, and concludes with a discussion of the study's significance and contribution to the existing body of knowledge.

1.1 Background of the Study

As computer technology advances, computers have entwined in almost every part of our daily lives. They became an integral part of our workplaces, schools, and homes. In response to technological advancements, a growing demand has emerged for computer programs that can enhance the quality of human life. Despite rising demand for computer programs, programming courses continue to exhibit high failure rates (Robins et al., 2003; Watson & Li, 2014). Teaching computer programming to beginners in various disciplines has been identified as a challenging task (Abdul-Rahman & Du Boulay, 2014). The difficulty of acquiring knowledge and skills necessary for programming computers has been frequently mentioned in the related literature on teaching computer programming (Blanco et al., 2009; Hawi, 2010; Kelleher & Pausch, 2005; Pokorny, 2009; Schulte & Bennedsen, 2006; Thomas & Greene, 2011; Watson & Li, 2014; White & Ploeger, 2004).

Teaching and learning programming presents a complex challenge for instructors and students. This phenomenon remains a significant challenge in the field of computer science education. Consequently, a substantial body of literature has investigated ways to improve computer programming education to enhance novice

students' learning of programming skills more efficiently (e.g., Abdul-Rahman & Du Boulay, 2014; Caspersen & Bennedsen, 2007; Coleman & Nichols, 2011; Haden et al., 2016; Harms, 2013; Looker, 2021; Van Merriënboer & Krammer, 1987). Within introductory programming courses, novice learners confront multifaceted learning challenges. They must concurrently acquire novel programming concepts, navigate unfamiliar development tools (e.g., Integrated Development Environments), and adapt to a paradigm shift in their problem-solving approaches (Mayer, 1987, as cited in Bucks & Oakes, 2011). These fundamental skills, coupled with their lack of familiarity with programming structure, design, and programming language syntax, present a significant challenge for many students (Baist & Pamungkas, 2017). Furthermore, programming concepts often appear abstract and lack a readily apparent connection to real-world phenomena. This disconnection hinders students' ability to link these concepts with their prior knowledge and experiences, which are crucial for meaningful learning (Myers, 1986).

Despite the established challenges in teaching programming to older age groups, such as university students, programming is now being introduced to younger children under the name of "coding." Recognizing the importance of early exposure to computational thinking and programming skills, many countries have integrated computer science and coding-related competencies into education curricula, including those for younger age groups (Fluck et al., 2016; Webb et al., 2017). In contrast to traditional text-based programming languages, which can be challenging for younger learners, a new trend in coding education is emerging with the use of graphical programming environments and block-based programming approaches. To make programming more accessible and engaging for children, graphical programming environments and block-based programming languages, such as code.org, Scratch and Tynker, have been used in their instruction (Grover & Pea, 2013; Resnick et al., 2009). These languages have many advantages over traditional text-based programming since they enable users to use graphical representations for programming (Koray & Bilgin, 2023; Meerbaum-Salant et al., 2013; Yang & Lin, 2019).

Teaching a subject that is already challenging for older groups to younger children will undoubtedly present its own set of difficulties. Research has identified various factors affecting programming learning, such as cognitive load (Grover & Basu, 2017), previous experience (Hinckle et al., 2020), mathematical background (Bennedsen & Caspersen, 2005; Bergin & Reilly, 2006; Grover et al., 2015; Mathews, 2017; Nasution et al., 2022), reading comprehension skills (Grover et al., 2016; Ma et al., 2023), misconceptions about programming (Grover & Basu, 2017), self-efficacy (Ketenci et al., 2019; Kinnunen & Simon, 2011; Kukul et al., 2017; Toma & Vahrenhold, 2018), gender (Beyer et al., 2003; Cheryan et al., 2015), socioeconomic status (Akpomudjere, 2020; Marks et al., 2006), learning styles (Abdul-Rahman & Du Boulay, 2014), learning goals (Hazley et al., 2015; Shell et al., 2013), and attitude (Ching et al., 2019; Sun et al., 2022).

Examining programming learning from a cognitive perspective is necessary for building a more generalizable understanding of effective programming education practices. Within this framework, cognitive load theory plays a significant role, as existing research studies suggest that a predominant challenge in programming lies in decomposing a problem into its constituent elements and articulating these components as programming code. Programming heavily relies on working memory, which exhibits limited capacity for storing and processing items, thereby potentially resulting in significant levels of cognitive load. Therefore, assessing the cognitive load encountered by students throughout the educational process is critical, given the multifaceted nature and inherent challenges associated with programming skill development (Berssanette & De Francisco, 2022).

Gender is another important variable that must be considered when examining factors influencing success in computer science. Despite increased female participation in Science, Technology, Engineering, and Mathematics (STEM), the IT sector continues to be male-dominated. The literature on Computer Science consistently highlights gender as a persistent inequity in computer science (Luxton-Reilly, 2016). While the roots of this issue are complex, factors such as stereotypes, prior experience, and self-efficacy have been identified as contributing to this

disparity. Stereotypes portraying computer specialists as male have been linked to lower interest in programming among girls compared to boys (Master et al., 2016). Consequently, female students often exhibit lower levels of self-efficacy in computer science (Beyer, 2014; Doubé & Lang, 2012). Boys typically have more prior programming experience (Bruckman et al., 2002) which results in a more positive attitude towards programming (Beyer et al., 2003) and higher levels of achievement (Guzdial et al., 2014). Nevertheless, research findings on the gender gap in computer programming have been inconsistent. While some studies indicate no significant gender differences among students (Akinola, 2015; Bennedsen & Caspersen, 2005; Bruckman et al., 2002; Gunbatar & Karalar, 2018; Qian & Lehman, 2016), others reveal persistent disparities (Guenaga et al., 2021; Tellhed et al., 2022). Efforts to introduce coding into early childhood education aim to address these inequities by providing equal opportunities for all children to develop digital competencies, potentially leading to a more diverse and inclusive IT workforce.

Socioeconomic status, a complex construct encompassing income, education, occupation, and perceived social standing, is another factor contributing to disparities in educational outcomes among students. Moreover, the geographical location of a school, often closely tied to socioeconomic status, has been shown to influence a range of educational outcomes. The concept of geographic location, described by Bæck (2016, p. 436) as “well-documented, less researched”, encompasses far more than just physical space. The location of the school, whether rural, suburban, or urban, impacts factors such as accessibility, resource allocation, and overall learning environment, thereby exacerbating educational inequalities (Chand & Mohan, 2019). Particularly in the realm of programming education, a school's geographical location plays a crucial role in influencing students' access to technology. Rural or suburban schools may encounter challenges in providing students with equitable access to advanced technological resources compared to their urban counterparts. In rural schools, the absence of up-to-date hardware and software has posed significant challenges for administrators and educators in effectively implementing computer programming curricula (Agnello et al., 2019). Adequate

resources are crucial for engaging young learners in programming and facilitating their comprehension of the subject matter (Yusof et al., 2021). Previous research has recorded teachers' complaints about the insufficiency of teaching resources (Greifenstein et al., 2021). Furthermore, research studies have shown that early exposure to digital technology (Gerson et al., 2022) and prior coding experience (Bowman et al., 2019; Bruckman et al., 2002; Grover et al., 2016) have an impact on programming learning. These factors are often mediated by socioeconomic variables such as social class, income level, and parental attitudes toward programming (Gerson et al., 2022), which could be indirectly linked to the geographical location of schools.

Motivational factors have also been highlighted in the literature as influential in programming education. The relationship between children's programming learning and motivational factors such as self-efficacy, attitude, and goal orientation is multifaceted and significant. The literature has demonstrated a positive relationship between self-efficacy, academic achievement, and performance (Bergey et al., 2015). Self-efficacy is a critical motivational construct that influences an individual's effort, resilience, and perseverance when faced with challenges (Bandura, 1977). In the context of programming education, students may encounter difficulties with complex programming and algorithmic problems. Individuals with low self-efficacy are likely to exhibit less perseverance in overcoming these obstacles, potentially hindering their progress and success in learning programming (Kovari & Katona, 2023; Ramalingam et al., 2004). While self-efficacy plays a significant role in influencing performance in computer programming courses, self-efficacy is also intricately linked to and influenced by other coding-related factors, including attitudes (Kovari & Katona, 2023), enjoyment (Kanaparan et al., 2017), prior programming experience (Ramalingam et al., 2004), as well as interest in computer science (Beyer, 2014). In addition to self-efficacy, attitude toward programming plays a pivotal role in student motivation and learning outcomes. However, the relationship between young learners' attitudes toward programming and their subsequent programming achievement remains inconclusive. While some studies

have suggested a significant correlation, others have found no such link (Baser, 2013). Given the relatively nascent nature of research on young children's attitudes toward programming, many studies have relied on broader measures of STEM attitudes (Ober et al., 2024). However, emerging research on the attitudes of middle school students toward computer science and programming suggests that these attitudes may be crucial in shaping long-term career aspirations, particularly in STEM-related fields.

Another fundamental component of motivation is the goals that are influenced by both individual and contextual factors. According to the goal orientation theory, students establish a range of goals to guide their performance across various academic tasks, including assignments, examinations, laboratory work, and overall course engagement (Elliot et al., 2011; Senko et al., 2011; Shell & Soh, 2013). Research indicates that adopting a goal-oriented approach to learning computer science enhances academic performance and student persistence (Shell et al., 2016). Generally, mastery-oriented and task-oriented goals exhibit a positive correlation with academic performance, whereas the impact of performance goals on course outcomes can be either adaptive or maladaptive (Elliot et al., 2011; Hazley et al., 2015; Hulleman et al., 2010; Tomić et al., 2020). However, students often have a complex interplay of multiple achievement goals, each influencing their motivation and behavior differently. Therefore, it is more accurate to adopt a comprehensive perspective rather than attributing a singular achievement goal orientation to them (Peteranetz, 2021).

Classroom goal structures are an integral component of goal orientation theory. In classroom settings, the types of goal orientations adopted by students are influenced by their perceptions of classroom goal structures (Midgley & Urdan, 2001). Studies have established that a perceived classroom mastery goal structure predicts personal mastery goal orientation and is positively associated with academic performance across multiple disciplines, such as mathematics (Guo & Hu, 2022; Urdan & Midgley, 2003). Conversely, a competitive classroom environment that emphasizes grades and social comparison tends to foster performance-oriented goals among

students (Meece et al., 2006). Therefore, the structure of classroom goals significantly impacts the behavior of students and their process of learning by influencing the types of personal goals students establish. Students' academic behaviors and strategies are influenced by both personal achievement goals and perceived classroom goal structures. Research findings indicate a positive relationship between self-handicapping behaviors and personal performance-avoidance goals. Moreover, classroom environments emphasizing performance-oriented goals have been associated with a rise in student engagement in self-handicapping strategies (Midgley & Urdan, 2001). Conversely, personal mastery goals and perceived classroom mastery goal structures have demonstrated weaker or negligible relationships with self-handicapping (Urdan et al., 1998).

Furthermore, with regard to goal orientation, cheating is another important concept studied in the literature. Studies have indicated that individuals with performance-oriented goals are more prone to engaging in behaviors associated with plagiarism than those with mastery-oriented goals (Anderman & Midgley, 2004). In computer science (CS) education, the majority of research on cheating behaviors has centered on higher education institutions and programming environments that are based on text. Particularly, studies have extensively examined cheating incidents within online programming courses and take-home assignments, where teachers have more difficulty monitoring students and students can easily share information and codes via the Internet (Abou Naaj & Nachouki, 2023; Hellas et al., 2017; Kim & Lee, 2022). Studies have found that plagiarism and cheating behaviors are associated with lower academic achievement, a desire to surpass peers, and the fear of failing. Furthermore, these behaviors have been shown to be less prevalent among female students (Abou Naaj & Nachouki, 2023; Newstead et al., 1996). This study investigated cheating behaviors among middle school students during in-class activities and collaborative learning sessions. However, the relevant literature suggests that students may misinterpret collaborative learning as an opportunity for dishonest behaviors, such as cheating, copying, or collusion (Barros et al., 2021). Furthermore, among the various factors influencing programming learning,

academic background emerges as a particularly significant predictor of success. Mathematics, in particular, has long been recognized as a strong predictor of programming achievement (Bennedsen & Caspersen, 2005). Moreover, research has provided evidence of a correlation between reading comprehension and programming proficiency (e.g., Lopez et al., 2008).

In conclusion, in reference to related literature, a large proportion of students have failed to reach a sufficient level of proficiency in their first computer programming course or even after they have taken more than one programming course. This problem is common in computer science education in many countries despite numerous research studies attempting to improve programming education. Teaching programming to younger children is a relatively new phenomenon, necessitating a deeper exploration of the unique challenges and influencing factors in this age group. Research studies have identified various factors affecting programming learning. Therefore, it is essential to investigate these factors, specifically in the realm of younger learners, to comprehend their influence on the learning process and to develop effective instructional strategies tailored to their needs.

1.2 Purpose of the Study

This study undertakes a comprehensive investigation into the factors that contribute to the acquisition of fundamental computer programming concepts among fifth-grade students within the framework of middle education. The primary objective is to gain an understanding of these multifaceted elements and their collective impact on students' programming proficiency. Through an in-depth exploration of sociodemographic and educational background, affective learner characteristics, motivation, learning environment, and cognitive load, this research seeks to unveil the dynamics inherent in the interplay of these factors.

Study variables encompassed a range of variables, including personal achievement goal orientations (mastery goal orientation, performance-approach goal orientation,

and performance-avoid goal orientation), perceived classroom goal structures (classroom mastery goal structure, classroom performance-approach goal structure and classroom performance-avoid goal structure), academic-related perceptions, beliefs and strategies (academic efficacy, academic self-handicapping strategies and cheating behavior), and attitude toward coding education. Additionally, the cognitive load was examined through the intrinsic, extraneous, and germane load. Educational background variables included mathematics and reading comprehension achievement, while sociodemographic background was assessed by examining the students' gender and the geographical location of their schools (urban vs. suburban). By addressing these factors, the study seeks to offer an in-depth understanding of the determinants in teaching programming to younger students, thereby contributing to the advancement of computer science education.

1.3 Research Questions

The main research question that guided this investigation was: *‘What factors influence the acquisition of fundamental computer programming concepts in fifth-grade students?’* This overarching research question serves as the focal point for investigating the complex relationships and interactions among various factors influencing students' acquisition of computer programming skills. The study will address this main research question through a detailed examination of the sub-research questions listed below:

1. Is there a significant difference in cognitive load experienced by students across seven fundamental programming topics?
2. Is there a significant difference in students' PALS (personal achievement goal orientations, perception of classroom goal structures, academic-related perceptions, beliefs and strategies), attitudes towards coding education, achievement in mathematics, achievement in reading comprehension, achievement in coding and cognitive load scores based on their gender?

- a. Is there a significant difference in students' PALS scores based on their gender?
 - b. Is there a significant difference in students' attitudes toward coding education scores based on gender?
 - c. Is there a significant difference in students' mathematics scores based on gender?
 - d. Is there a significant difference in students' reading comprehension scores based on gender?
 - e. Is there a significant difference in students' coding achievement scores based on gender?
 - f. Is there a significant difference in students' cognitive load scores across seven fundamental programming topics based on gender?
3. Is there a significant difference in PALS (personal achievement goal orientations, perception of classroom goal structures, academic-related perceptions, beliefs and strategies), attitudes towards coding education, achievement in mathematics, achievement in reading comprehension, achievement in coding, and cognitive load scores between students from urban schools and suburban schools?
- a. Is there a significant difference in PALS scores between students from urban schools and suburban schools?
 - b. Is there a significant difference in attitudes toward coding education scores between students from urban schools and suburban schools?
 - c. Is there a significant difference in mathematics scores between students from urban schools and suburban schools?
 - d. Is there a significant difference in reading comprehension scores between students from urban schools and suburban schools?
 - e. Is there a significant difference in coding achievement scores between students from urban schools and suburban schools?

- f. Is there a significant difference in cognitive load scores across seven fundamental programming topics between students from urban schools and suburban schools?
4. How do research variables predict students' achievement scores in programming?
5. What are the students' experiences and opinions on the factors that affect their learning fundamentals of programming?

Examining these sub-research questions intended to gain insights into the specific aspects of learner characteristics that may impact students' coding performance and, ultimately, enhance the overall comprehension of the factors influencing their academic achievement in computer programming.

1.4 Significance of the Study

The teaching and learning of computer programming present persistent challenges, documented by high failure rates across various levels, courses, and teaching contexts (Abdul-Rahman & Du Boulay, 2014; Watson & Li, 2014). There have been many research studies that aim to explore different course design approaches, tools, and instructional strategies that facilitate learning the computer programming process by helping students acquire the required knowledge and skills. Some attempts result in substantial success while others do not, but still, the leading reasons behind students' success or failure in programming are not fully understood. This study aimed to address this gap through a multifaceted lens by examining the factors influencing students' achievement in fundamental programming skills.

This study goes beyond cognitive factors by incorporating motivational variables derived from goal orientation theory, which emphasizes the substantial influence of student goals on academic achievement (Meece et al., 2006). The intricate relationship between affective, motivational, and environmental factors and their impact on students' achievement in the context of programming remains

inadequately understood. Traditionally, programming education research has focused on undergraduate or graduate-level fields such as engineering, mathematics, and computer science, where introductory programming has fundamental importance for any student. However, the recent emphasis on introducing programming to younger learners necessitates a shift in research focus. While many studies implement innovative learning tools and methodologies for younger students, they often lack a deep understanding of the complex and interrelated challenges in programming instruction and learning. These challenges encompass not only instructional approaches but also students' cognitive abilities, motivational factors, and the learning environment. Considering the multitude of variables affecting learning in a broad sense and the specific domain of learning programming, there is a compelling need to comprehensively evaluate factors influencing students' achievement in the fundamentals of computer programming, encompassing both cognitive and motivational aspects.

Building upon the established challenges in teaching computer science and the growing trend of introducing programming concepts to younger children, this study holds significant practical value for educators, curriculum developers, and policymakers. By investigating the multifaceted determinants that influence middle school students' acquisition of fundamental programming principles, the results can contribute to the development of enhanced instructional strategies tailored to this age group. Given the complex nature of computer programming, which demands proficiency across multiple domains and is widely recognized as challenging to acquire, it is imperative to consider the cognitive load placed on learners and their capacity to process information during instruction (Berssanette & De Francisco, 2022). Understanding the interplay between sociodemographic background, educational experiences, learner characteristics, motivational factors, learning environment, and cognitive load can equip educators with a comprehensive framework to respond to the varied needs of students in their classrooms. This knowledge can be translated into the creation of differentiated learning experiences

that address various learning styles and abilities, fostering a learning environment that is more inclusive and interactive for computer science education.

The findings of the study can make a valuable contribution to curriculum development by providing a profound understanding of the appropriate level of complexity and the specific programming concepts most conducive to successful learning for middle school students. This information can guide the creation of age-appropriate and engaging curriculum materials that stimulate a desire to develop proficiency in coding and equip young learners with the foundational knowledge and abilities required for thriving in the era of digital advancements. Ultimately, by identifying the factors that shape programming achievement within this age group, this study aims to contribute valuable insights that can inform educational practices and pave the way for more effective pedagogical approaches and instructional strategies for teaching computer science to younger learners.

CHAPTER 2

LITERATURE REVIEW

This chapter serves as a theoretical foundation for contextualizing and framing the research questions through a detailed examination of existing research. The primary aim is to reveal a complete comprehension of the theoretical underpinnings of cognitive load theory and motivational factors, particularly regarding the context of computer programming education for children.

2.1 Cognitive Load Theory

Cognitive Load Theory (CLT) posits that the capacity of working memory to process information is a critical factor in learning outcomes. This theory emphasizes the importance of optimizing cognitive resources to enhance learning efficiency. To investigate how CLT can be applied to enhance computer programming instruction, the fundamentals of CLT, including the architecture of human cognition, core principles of the theory, and the classification of cognitive load were examined. Furthermore, the role of CLT in shaping effective programming instruction was explored.

2.1.1 Human Cognitive Architecture

Human cognitive architecture provides insight into the processes of learning, thinking, and problem-solving. This architecture, resembling a natural system for information processing, employs various strategies to manage cognitive load. A key focus of cognitive load theory is to identify strategies for decreasing this load, thereby facilitating the transition of information from working memory to long-term memory, where biologically secondary knowledge resides. The following sections

provide an in-depth examination of the role of information categories, long-term memory, working memory, and schema theory within this framework.

2.1.1.1 Categories of Information

A diverse array of classification schemes exists for knowledge, with one notable classification distinguishing between biologically primary and biologically secondary knowledge. This distinction is substantial because the acquisition, organization, and storage of various categories of knowledge may require different instructional methodologies.

From Geary's (2008) evolutionary perspective, biologically primary knowledge pertains to information that humans have naturally developed the capacity to acquire. In contrast, biologically secondary knowledge encompasses information that has not been naturally acquired through human evolution and has become essential due to cultural influences (Sweller et al., 2011). Biologically primary knowledge encompasses abilities and skills that humans have developed through natural selection. They encompass the differentiation of facial features and vocal patterns, the utilization of general problem-solving strategies, and fundamental rudimentary social interactions. Competencies grounded in biologically primary knowledge are typically acquired automatically and, more frequently, subconsciously, devoid of formal educational intervention. For example, we are not taught how to talk via any curriculum since we have evolved to acquire this skill (Sweller, 2016).

Biologically, primary knowledge serves as the foundation for the majority of human cognition. Although biologically primary knowledge is essential to human cognition, it does not directly lead to intelligent behavior. However, for complex problem-solving and reasoning, biologically primary knowledge must be integrated with and built upon biologically secondary knowledge. The cognitive processing required for biologically evolved primary abilities differs from the typical information processing required to obtain biologically secondary knowledge. For instance, cognitive

processing for learning to speak, which requires biologically primary knowledge, is not the same as that for learning to write, which requires biologically secondary knowledge (Geary, 2008; Sweller et al., 2011).

Instructional design primarily focuses on biologically secondary knowledge, encompassing activities such as reading, writing, and other subjects taught through instruction (Cowan, 2014; Sweller, 2016). However, it is important to recognize the value of biologically primary knowledge. Since acquiring primary knowledge requires minimal conscious cognitive resources, a strategic approach could involve utilizing these inherent abilities as a scaffolding for learning secondary knowledge. In other words, instructional design can leverage the efficiency of primary knowledge to minimize the load on working memory associated with acquiring complex skills and concepts (Paas & Sweller, 2012).

2.1.1.2 Long-Term Memory

Long-term memory serves as a vast storage for knowledge and skills that we retain in a way that allows for more or less permanent access. All the things we "know," such as our name, the alphabet, reading, writing, and swimming, are retained within our long-term memory, ready to be accessed whenever needed (Cooper, 1998).

A valid argument may be made that a substantial portion of the data retained in long-term memory comprises biologically primary knowledge (Sweller, 2020). Many tasks that we often perceive as effortless and uncomplicated are facilitated by our primary knowledge because all biologically primary skills are perceived as easy and simple. Conversely, activities such as chess, which rely on secondary knowledge that has not been naturally acquired through evolution, are seen as exceedingly difficult. This distinction between primary and secondary knowledge highlights the crucial role of long-term memory in facilitating higher-level cognitive processes. While primary knowledge allows for seemingly effortless tasks, long-term memory plays a

much more substantial role in human cognition, particularly in areas like thinking and problem-solving (Sweller et al., 2011).

Long-term memory is essential for all higher-level cognitive processes, including thinking and problem-solving (Bliss & Collingridge, 1993). It is critical not only for human cognition but also for those facets of cognition considered the highest levels of human intelligence. The role of long-term memory goes beyond facilitating the retrieval of past events, whether meaningful or not (Kandel et al., 2014). The impact of long-term memory on cognition and the nature of the cognitive changes that resulted from practice were revealed by investigations conducted by De Groot (1978) and Chase and Simon (1973). In his research with chess players, De Groot (1978) identified only one distinction that separated players of different skill levels, and it appeared unrelated to their problem-solving proficiency. Instead, De Groot's (1978) focus revolved around the concept of memory. In a parallel study, Chase and Simon (1973) arranged the chess pieces in a random configuration. No distinctions were observed between chess players with different levels of proficiency with regard to random configurations. The findings of these two research studies hold the potential to provide a comprehensive understanding of chess expertise in a manner that does not require consideration of additional variables. Chess proficiency does not primarily revolve around thinking skills; instead, it is derived from the capacity to recognize numerous configurations for chess boards and identify the optimal strategies for each. When chess players have no knowledge stored in their long-term memory, they are required to formulate strategies through problem-solving searches. While thinking skills are necessary for novices, experts require fewer problem-solving searches as their expertise increases (Sweller et al., 2011). Explanations offered to distinguish the cognitive processes between expert and novice chess players can be expected to be extrapolated to encompass all processes demanding the utilization of secondary knowledge (Sweller, 1988). From this perspective, possessing information about specific problem situations and their accompanying moves should be the primary factor in improving problem-solving skills rather than acquiring general problem-solving strategies (Gilhooly & Green, 1988). In essence,

these findings suggested that for complex domains requiring secondary knowledge, a focus on building a rich knowledge base of specific examples and solutions may be more effective than generic problem-solving approaches (Sweller et al., 2011).

2.1.1.3 Working Memory

Working memory is a construct introduced by psychologists in the mid-20th century. It refers to the system responsible for temporarily holding and manipulating information. Working memory holds significant theoretical weight in psychology, particularly because it highlights the limitations of our cognitive capacity. Early discussions hinted at working memory being essential for our everyday lives, especially when it comes to planning tasks. The concept later broadened to encompass the mental ability to remember plans in general, not just those related to daily routines. This cognitive system with limited capacity is crucial for the temporary retention and active information management in a readily accessible state. This processed information plays a critical role in facilitating higher-order cognitive processes such as planning, comprehension, reasoning, and problem-solving (Cowan, 2014).

Over the years, debates have emerged about the specific limitations of working memory. Some key questions include its capacity, processing speed, duration of retaining information, and interference properties. While working memory capacity can hold around seven chunks of information (Miller, 1956), the ability to actively process that information is more limited. Working memory capacity is estimated to be closer to two to three chunks for tasks requiring manipulation, such as organizing, contrasting, or comparing information. Interactions among elements maintained within the working memory itself consume additional working memory resources. This effectively reduces the number of unique elements that can be actively processed concurrently (Sweller et al., 1998). This number falls far short of the complex interactions that occur in most intellectual endeavors. Working memory alone would only permit rudimentary cognitive activities (Paas et al., 2003).

However, by leveraging long-term memory and established knowledge structures (schemas), more complex tasks could be handled more effectively.

2.1.1.4 Schema Theory

Schema theory posits the existence of cognitive constructs called schemas stored in long-term memory (Rumelhart & Norman, 1976). These schemas act as fundamental units or chunks of knowledge, categorizing information about the world around us. Schemas play a crucial role in information processing by influencing how we perceive and interpret new experiences. However, these frameworks are personal, and individual experiences can influence how schemas are formed and modified. In the process of accretion, a new experience seamlessly integrates into an existing schema without causing substantial modifications to the schema. Any incoming information perceived as aligning with a particular schema will be processed in a consistent manner, facilitating efficient storage and retrieval (Sweller et al., 2011). This is because new knowledge is often assimilated into existing schemas during learning, leveraging established connections. Conversely, tuning occurs when a novel experience challenges the existing schema, leading to the adaptation of the structure to integrate new information. Finally, when a new experience deviates significantly from existing frameworks, a process of restructuring takes place, leading to the creation of a new schema (Rumelhart & Norman, 1976).

In addition to facilitating the organization and storage of knowledge, schemas also have a critical function in reducing the cognitive load on working memory. Through continuous learning experiences, schemas may encompass extensive amounts of information. Serving as a framework for interpreting new information based on existing knowledge structures, schemas effectively reduce the load placed on working memory (Van Merriënboer & Sweller, 2010). This allows for the distribution of these limited working memory resources to more complex cognitive tasks. Over time, schemas develop into complex networks of information through a cumulative process of continuous addition, combination, and rearrangement. This

evolution enhances knowledge representation and processing efficiency. Automation plays a significant role in this development. In contrast to conscious processing, automatic processing, arising from extensive practice, significantly reduces the reliance on working memory. Constructed schemas can become automated through repeated application. From an instructional design standpoint, effective instruction should aim to promote both schema construction and schema automation for consistent aspects of tasks across various problems (Sweller et al., 1998).

2.1.2 Foundations of Cognitive Load Theory

Cognitive load (CL), proposed by Sweller in the 1980s, refers to the amount of cognitive resources allocated to working memory that a learner is expected to dedicate to processing new information (Berssanette & De Francisco, 2022). This theory is grounded in the well-established model of human cognitive architecture, which encompasses both working memory and long-term memory. Cognitive Load Theory (CLT) posits that human cognitive processing is heavily influenced by the constraints on the capacity and duration of working memory. Since newly acquired information necessitates initial conscious processing within limited capacity and short-duration working memory, this can hinder learning effectiveness (van Merriënboer & Sweller, 2005).

Expertise emerges gradually as individuals progressively integrate simpler concepts into more complex schemata. These schemata organize and store knowledge for efficient processing, thereby reducing the load on working memory. This is because even highly complex schemata can be treated as single units within working memory, reducing the number of independent elements requiring processing (van Merriënboer & Sweller, 2005). Conversely, working memory capacity exhibits limitations when encountering novel information. This stems from the absence of pre-existing schemata, which act as a central executive, facilitating the processing and organization of familiar concepts. Furthermore, the challenge is compounded by

the exponential growth in complexity as the number of elements within novel information increases linearly. Unlike readily organized knowledge integrated into schemata, novel information lacks a pre-established framework, imposing a load on working memory's limited processing capabilities (Sweller et al., 1998).

Contrary to prevailing beliefs and several cognitive theories, CLT asserts that specific forms of problem-solving activities may impede the learning process. Sweller (1988) underscored, in summarizing the findings of his study, that conventional problem-solving may not contribute to schema acquisition, given the substantial differences in the mechanisms required for problem-solving and schema acquisition. As schema acquisition represents a critical element of problem-solving expertise, an excessive emphasis on the problem-solving process may place a high level of cognitive load and impede the development of expertise. Although schemas serve as a template that simplifies complex problem-solving tasks, conversely, they might complicate the resolution of simple problems when erroneously assuming the schema's relevance to the problem and its provision of an appropriate template. Utilizing an inappropriate problem-solving schema resulted in *Einstellung*, also recognized as a mental set that obstructs our perception of apparent details (Sweller et al., 2011). Utilization of complicated problem-solving techniques, such as means-ends analysis, might result in an increased cognitive load on the learner, potentially resulting in a more pronounced hindrance to the learning process. Engaging in problem-solving tasks that require storing many items in short-term memory may lead to an excessive cognitive load (Sweller, 1988).

Cognitive load (CL) denotes the extent of resources allocated to working memory, anticipated for a learner to allocate in order to process new information. With this approach, CLT, which serves as an instructional design theory, attempts to explain the impact of the information processing load generated by learning tasks on learners' capacity for information processing and constructing knowledge in long-term memory (BerSSanette & De Francisco, 2022). Chandler and Sweller (1994) asserted that learning may be more difficult from materials that include large amounts of information than learning from materials that have less information.

When students are exposed to excessive amounts of information and inadequate regulation of the complexity of instructional materials, this will lead to an excessive cognitive load due to the constrained capabilities of short-term memory.

Different instructional strategies and media are used in instruction. According to CLT, their effectiveness cannot be ensured if the cognitive architecture of the brain is not considered during instruction. Information is retained within the long-term memory in a vast number of schemas. Schemas organize categories and elements of data, as well as the relationships among them (Chi et al., 1982). These schemas are constructed in working memory. For highly skilled performance, exceedingly complicated schemas, which incorporate “elements consisting of lower-level schemas into higher-level schemas”, are constructed (Paas et al., 2004, p.2). But, as a consequence of the limitations of working memory capacity, dealing with this kind of complex schemas may exceed working memory capacity. However, due to comprehensive and sufficient practices, schemas can be automated. The automation of those schemas allows them to be processed unconsciously, thereby decreasing the working memory load.

The primary focus of cognitive load theory is to manipulate working memory load in a way that facilitates the construction of schema within the long-term memory for automation of schemas, which results in learning. CLT is concerned with the ease of processing information within working memory. The cognitive load imposed on working memory can be influenced by two key factors: the inherent complexity of the learning tasks themselves (intrinsic cognitive load) and the way these tasks are presented (extraneous cognitive load) (van Merriënboer & Sweller, 2005).

2.1.3 Types of Cognitive Load

Traditional cognitive load theory delineates three primary types of cognitive load: intrinsic, extraneous, and germane. Intrinsic cognitive load, defined as the load that stems from the inherent complexity of learning materials and the learner's expertise

level, is considered independent of the educational methods implemented (Sweller et al., 2011). This independence arises because the number of elements requiring simultaneous processing in working memory depends on the degree of element interactivity within the learning materials or tasks. Understanding materials with highly interactive elements presents a significant challenge. However, the key to fostering comprehension lies in developing cognitive schemata that integrate these interacting elements. Consequently, what a novice learner perceives as a large number of interacting elements may be a single element for a more experienced learner with a well-developed schema (van Merriënboer & Sweller, 2005).

Extraneous cognitive load arises from the tasks learners perform or the way information is presented. Within this type of cognitive load, the instructional design employed to deliver the material may additionally impose a load that is extraneous and unrelated to the intended learning objectives (Sweller et al., 2011). In contrast to intrinsic cognitive load, extraneous cognitive load is not essential for the process of knowledge acquisition, such as schema construction and automation. Unlike intrinsic load, extraneous cognitive load can be effectively reduced or eliminated through the implementation of targeted instructional interventions (van Merriënboer & Sweller, 2005).

Intrinsic and extraneous cognitive load have an additive effect on the learner's working memory. Thus, the impact of extraneous cognitive load on student learning outcomes depends on the intrinsic cognitive load. In situations where the intrinsic load is elevated, instructional design should prioritize minimizing extraneous cognitive load. This ensures that the overall cognitive load is kept at a level that aligns with the limitations of working memory and facilitates successful knowledge acquisition. Conversely, for tasks with a lower intrinsic load, a moderate level of extraneous cognitive load, even if stemming from instructional design shortcomings, may not necessarily impede learning. However, it is important to acknowledge that even in low intrinsic load situations, excessive extraneous cognitive load can still hinder learning by exceeding working memory limitations (van Merriënboer & Sweller, 2005).

In earlier versions of CLT, the germane load (GL) was introduced by Sweller et al. (1998) as an additional beneficial load on the learning process. This construct aimed to explain the positive impact of certain variations within learning materials on the learner's cognitive processes during knowledge acquisition. Research studies have demonstrably shown that variability in practice activities while increasing cognitive load, facilitates schema construction and enhances the transfer of training. Paas and Van Merriënboer (1994) proposed that the observed increase in cognitive load stemmed from processes directly contributing to learning, such as automation and schema construction, rather than from extraneous cognitive load that does not promote learning. The introduction of germane load in CLT emphasizes the importance of active learner engagement for successful learning. While CLT focuses on managing cognitive load, germane load underlines the necessity for learners to invest mental effort in processing information relevant to building knowledge and schemas. This processing requires some level of motivation and willingness to engage with the learning materials. While instructional strategies can be employed to minimize extraneous load and free up cognitive resources, this approach is only effective if learners are motivated to invest these freed resources in germane cognitive activities that promote schema development and knowledge acquisition. In essence, effective learning depends on optimizing instructional design to reduce extraneous load and foster learner motivation to engage in germane cognitive processing actively (van Merriënboer & Sweller, 2005).

While traditional conceptualizations of CLT involve the three types of cognitive load, as previously outlined, recent years have seen growing debate regarding the germane load construct. Research studies within CLT have emphasized that cognitive load does not consistently hinder the process of learning. In fact, it is essential for facilitating meaningful learning, particularly complex learning, which necessitates effortful cognitive processing and the load that is linked with working memory. However, when CLT was first introduced, intrinsic and extraneous loads were considered loads to be properly managed or minimized to prevent cognitive overload. Consequently, the germane load construct was introduced to reflect the

purposeful cognitive effort invested in learning and the associated demands placed on working memory during knowledge acquisition. Unlike intrinsic and extraneous cognitive load, the construct of germane load was introduced within the CLT framework based primarily on theoretical considerations. While the additive hypothesis, which posits a cumulative effect of the three load types on learners, provided initial support for the framework, empirical evidence specifically demonstrating the need for germane load remains a topic of ongoing discussion (Greenberg & Zheng, 2023; Kalyuga, 2011). However, over time, the limitations of this additive framework have begun to be discussed by the researchers as it does not fully capture the complex interactions between these loads. Critics argue that germane load and intrinsic load are not entirely distinct. They contend that germane load inherently relies on intrinsic load. Without the inherent challenge presented by the material (intrinsic load), there would be no need for the learner to exert effort in processing it (germane load) to build new knowledge (Greenberg & Zheng, 2023). Therefore, a debate was started on whether germane load should be regarded as a discrete form of cognitive load or a germane resource in working memory (Leppink & van den Heuvel, 2015). In their 2019 revisit of the topic titled "Cognitive Architecture and Instructional Design", Sweller et al. emphasized that the growing body of empirical research consistently demonstrated a key finding: reducing extraneous cognitive load led to a corresponding decrease in overall cognitive load. The new formulation of CLT presumes that germane cognitive load doesn't simply increase overall cognitive load. Instead, it reallocates working memory resources from irrelevant tasks to core learning activities, enabling efficient processing of task-intrinsic information.

There have been some research studies that provide evidence supporting a framework in cognitive load theory that emphasizes the role of germane resources in working memory on shaping learners' cognitive effort during learning. Findings from a study by Kalyuga (2011) supported the notion that the traditional CLT framework might be redundant. This study suggested that germane load might not be a distinct category but could potentially overlap with, or even be indistinguishable from,

intrinsic load. Similarly, findings from a recent study by Greenberg and Zheng (2023) suggested that germane load might not be a direct predictor of performance outcomes, while intrinsic load was the primary variable influencing performance outcomes. This finding led them to propose that mental activity directly relevant to learning (germane activity) might be more closely linked to the cognitive resources available in working memory rather than constituting a separate type of cognitive load.

Additionally, they found that individuals with greater working memory capacity could effectively manage complex learning tasks while still allocating cognitive resources to learning, proposing that the exertion of cognitive effort during the process of acquiring knowledge is influenced by germane resources stored in working memory, as opposed to the germane load. On the other hand, a meta-analysis of cognitive load questionnaires found that the concept of germane load continues to provoke debate with regard to its measurement and theoretical integration, supporting the need for further investigation (Krieglstein et al., 2022). In recent frameworks of cognitive load theory, the term “self-perceived learning” has been widely used to refer to germane load. Self-perceived learning emphasizes the learner's own perception of how much they are acquiring knowledge (Bergman et al., 2015; Fredericks et al., 2021; Quintero-Manes et al., 2022). Due to the ongoing research on the cognitive load imposed during learning, this study utilized the term "germane load" to address the germane resources, thereby avoiding potential conflicts.

2.1.4 Research on the Role of Cognitive Load Theory in Computer Programming Education

Computer programming represents a cognitively complex domain characterized by necessitating mastery across various competencies, which often leads to significant challenges for learners due to the high CL it imposes on their working memory (Berssanette & De Francisco, 2022). Working memory is critical for solving

problems like programming comprehension as it allows us to hold relevant information in mind, manipulate it, and make connections to reach solutions. The limitations of working memory pose a significant challenge in computer programming education. Cognitive Load Theory arises as a beneficial framework in computing education by addressing this challenge. One of the fundamental challenges, particularly in introductory computer programming education, lies in the subject matter's inherent complexity. This complexity generally stems from two factors: the number of new concepts students must learn and the requirement to integrate these concepts with previously learned ones to solve problems. From a perspective rooted in CLT, the process of integration increases the cognitive load encountered by students, thereby hindering their learning and problem-solving process. CLT sheds light on the impact of human cognitive architecture limitations on the process of learning and offers guidance for optimizing learning processes (Duran et al., 2022). Research studies have shown that the application of CLT in teaching computer programming involves various strategies, including applying concepts like the worked example effect, the development and use of instructional resources or tools, and pedagogical strategies based on measuring cognitive load.

Related literature has shown that instructional methods based on CLT impact the effectiveness of teaching computer programming. Effective learning in programming education can be achieved through methods that reduce extraneous load and optimize intrinsic and germane loads. Besides that, by leveraging CLT principles, educators could differentiate intrinsic from extraneous cognitive load. This distinction empowers them to design instructional strategies that effectively manage the learning process and optimize students' cognitive resources (Looker, 2021). One of the most widely employed strategies in computer science education (CSE) for aiding students' knowledge acquisition is worked examples (Abdul-Rahman & Du Boulay, 2014; Caspersen & Bennedsen, 2007; Derry, 2000; Garner, 2002; Gray et al., 2007; Hsu et al., 2012; Lim, 2019; Mason & Cooper, 2013; Muldner et al., 2022; Nainan & Balakrishnan, 2019; Sands, 2019; Takir, 2011). Worked examples, comprising a problem statement, solution steps, and the final

result, have demonstrated the enhancement of learning outcomes by reducing cognitive load, shortening learning time, and facilitating the construction of cognitive schemas. Consequently, this enables students to solve similar problems more with increased efficiency and effectiveness (Sweller & Cooper, 1985). While worked examples have been shown to be a valuable tool in programming education, particularly for novice learners encountering complex technical concepts for the first time, their effectiveness is influenced by their design and implementation. These examples may cover different problem types, programming paradigms, and visualization techniques to address a variety of learning styles and deepen understanding. Programming education utilizes a variety of design types for worked examples, including text-based static examples, modeling examples, dynamic code-tracing, animated examples, and incomplete examples (Muldner et al., 2022). Despite the benefits of worked examples, one potential drawback of them lies in their passivity; they may not inherently compel learners to engage in a meticulous analysis of the presented solution. To address this shortcoming, Van Merriënboer and Krammer (1987) introduced the utilization of completion problems in introductory computer programming education.

Empirical studies have demonstrated various innovative educational tools and techniques aimed at enhancing the instruction and comprehension of computer science. This emphasis on innovative approaches underscores the inherent challenges students face in grasping complex concepts within the field. In this context, CLT provides valuable insights into how to design these educational tools and techniques. One such tool is linked list visualization software designed based on Cognitive Load Theory's split-attention effect, which integrates diagrams and code. Results showed that this approach could help students with prior programming knowledge gaps visualize and understand linked lists more effectively, thereby reducing cognitive load and fostering a deeper grasp of data structures (Arevalo-Mercado et al., 2023). The findings of another study demonstrated that a new teaching method using a custom visualization tool helped novice programmers grasp function-based problem-solving in a visual setting. Compared to traditional teaching

methods, students using the new approach performed significantly better on tests and assignments (Winter et al., 2019). Another study, drawing on principles from CLT, proposed by Harms, (2013), suggested that developing personalized tutorials tailored to a user's programming expertise could enhance the effectiveness of novice programmers in learning new programming concepts encountered in unfamiliar code. The main purposes of the proposed tutorials were to predict the learners' potential cognitive load by modeling their expertise in programming and to minimize the extraneous cognitive load by presenting programming concepts that prevent exceeding the working memory capacity of a learner. According to the findings of the studies, by effectively managing cognitive load, these innovative approaches hold great promise for improving computer programming education.

2.2 Motivation and Learning

Numerous theories of motivation in learning have been developed through research from diverse perspectives, offering significant insights into the factors driving student engagement and achievement. Educators have access to an extensive array of resources related to student motivation. These theories provide essential understandings of the underlying sources of curiosity and persistence in learners. Among these, goal orientation theory examines how students' goals influence their motivation and learning behaviors, providing a framework for understanding student engagement.

2.2.1 Goal Orientation Theory

Goals are the cornerstone of human motivation, propelling us to strive for achievement and growth. Achievement Goal Theory offers insights into how individuals set goals within various contexts, including education. This theory, initially applied to understand young athletes, posits that their perceptions of ability (shaped by past achievements and comparisons with others) and their definitions of

successful outcomes significantly influence the types of goals they set for themselves (Nicholls 1984, 1989). These goals, in turn, shape their overall motivational processes and influence their training and performance behaviors.

Personal Achievement Goals

Goal orientation theory, building on achievement goal theory, proposes that students' motivation and learning behaviors are influenced by the types of goals they set for themselves. Numerous models of goal structure have been created to enhance understanding of the motivations driving achievement behaviors. Early research within achievement goal theory explored goal structure through two dimensions, focusing on mastery and performance goals. Individuals who possess mastery goals prioritize learning and improvement by focusing on acquiring knowledge and developing skills. Their success is measured by personal growth, not just achieving a specific grade. For mastery goal-oriented learners, the desire to learn and improve is their primary motivator, even when encountering difficult problems. On the other hand, performance goal orientation encompasses a range of goals focused on relative achievement. The goal of performance-oriented learners is to demonstrate competence and outperform others. The emphasis is less on mastery of the concept itself and more on their performance and how it compares to others (Ames & Archer, 1988; Elliott & Dweck, 1988).

Self-theories, such as Dweck's self-belief theory, further explain how students' perceptions of their abilities influence their goal-setting. For instance, students who possess a growth mindset tend to be more inclined to accept challenges and establish goals aimed at achieving mastery (Dweck, 1986). Subsequently, avoidance goals were incorporated into the framework, specifically as performance-avoidance goals (Skaalvik, 1997). Contrary to performance-approach goals, students with performance-avoidance goals are motivated by a concern about the possibility of not succeeding, focusing on avoiding negative outcomes rather than embracing challenges and aiming for improvement. These performance-avoidance goals are

generally less effective in driving academic achievement (Liem et al., 2008; Shell et al., 2013; Shell & Soh, 2013).

Elliot and McGregor (2001) sought to refine the understanding of mastery goals by proposing a 2x2 model. This model differentiates between mastery-approach goals and mastery-avoidance goals. Mastery-approach goals, similar to the original conceptualization, emphasize a desire for learning and improvement. However, mastery-avoidance goals, a recent addition to the framework, are driven by a fear of failing to master the task or knowledge. Critics of this new framework have argued that the 2x2 structure might be too complex and difficult to measure in real-world settings. Another version of this model was then proposed and tested by Elliot et al. (2011) as a 3x2 model, including three approaches and avoidance goals: task, self, and other. Furthermore, some researchers raise doubts about the presence of a distinct mastery-avoidance goal orientation, suggesting that it might conceptually overlap with performance-avoidance goals (Pintrich, 2000). Current research suggests a further refinement of performance goals by distinguishing between normative and appearance goals (Hulleman et al., 2010). Normative goals emphasize social comparison, motivating learners to outperform others or avoid underperforming relative to their peers. In contrast, appearance goals center on managing self-presentation, driving learners to either showcase their abilities or conceal their shortcomings from others. Normative goals appear less likely to lead to negative outcomes. However, appearance goals, with their focus on avoiding negative self-evaluation and potential public shame, might be more closely linked to maladaptive behaviors (Zingaro et al., 2018).

Classroom Goal Structures

The goal-oriented messages perceived by students in the classroom form the classroom goal structures. Analysis of the Patterns of Adaptive Learning Scales (PALS) instrument has revealed that personal achievement goal orientations and classroom goal structures are distinct constructs. However, related literature

indicates that perceived classroom goals act as predictors of personal achievement goals.

Perceptions of the learning environment focusing on effort and understanding are positively associated with the adoption of mastery-oriented goals by students (Meece et al., 2006; Turner et al., 2002; Urdan, 2004). Students who perceived a classroom environment that emphasized mastery goals reported employing more effective learning approaches, demonstrating a preference for more challenging tasks, and exhibiting a more positive disposition towards the learning environment (Ames & Archer, 1988). Research studies have discussed the influence of teachers on the formation of the classroom goal structure. The evaluation strategies or group strategies employed by teachers significantly impact how these structures are formed. For instance, creating ability groups or employing evaluation strategies that foster a competitive climate in the classroom can strengthen perceived performance-goal structures (Meece et al., 2006). Soltani et al. (2022) further revealed that students' perceptions of competition, the perceived significance of the subject matter, and their personal orientation towards mastery goals all positively contributed to their academic performance. While performance-approach goals can be beneficial for students, their effectiveness may vary depending on factors such as gender, age, and the learning environment. Research conducted by Midgley et al. (2001) suggested that these goals may be more advantageous for boys compared to girls and for older students compared to younger ones. Additionally, the presence of mastery goals alongside performance-approach goals may further enhance these benefits, particularly in competitive learning environments.

2.2.2 Self-efficacy

Self-efficacy, a concept central to Bandura's Social Cognitive Theory, refers to a learner's belief in their capabilities to master knowledge and skills. This belief acts as a cornerstone of motivation, directly influencing how much effort and perseverance a learner is willing to invest (Bandura, 1977). Several factors shape this

crucial belief, including previous success or failure experiences, social persuasion, emotional states, and observational learning.

Individuals use various information sources to evaluate their capabilities. Performance accomplishments, referring to past successes and failures, are considered the most reliable source of information for self-efficacy appraisal. They provide concrete evidence of one's capabilities, directly demonstrating what one can accomplish. Overcoming challenges builds confidence, while repeated failures can undermine self-belief. Furthermore, indirect experiences, such as observing others succeed in similar tasks, can boost confidence in one's own abilities. Conversely, witnessing failures can lower self-efficacy. Social influences and interactions also play a significant role in self-efficacy. Social persuasion, which refers to positive encouragement, commendation, and expressions of confidence from others, strengthens an individual's confidence in their capabilities. Learners who receive encouragement and support from peers, teachers, or mentors are more likely to develop strong self-efficacy. For example, teacher acknowledgment (environmental) reinforces students' perception of progress (personal), fostering intrinsic motivation and self-efficacy to support ongoing learning. Finally, physiological and emotional states encountered during the learning process, such as anxiety or confidence, also play a role in self-efficacy appraisal. Individuals experiencing lower anxiety in a situation may interpret this as a sign of greater capability, while higher anxiety levels might be perceived as indicating lower competence (Bandura, 1977; Schunk & DiBenedetto, 2020; Usher, 2009).

2.2.3 Academic Self-Handicapping Strategies

Students who are concerned about failing exams or assignments may perceive their self-esteem to be at risk. A common coping mechanism for this concern is the use of self-handicapping strategies (Schwinger et al., 2014). Academic self-handicapping strategies involve behaviors in which students create impediments either before or during the task they need to accomplish, thereby hindering their success. Examples

of academic self-handicapping behaviors exhibited by some students include procrastinating and spending excessive time on other activities, such as socializing, thus leaving them very little time to study. The purpose of these strategies is to provide an excuse for failure so that the failure can be attributed to these self-imposed impediments rather than to a lack of ability (Urdan, 2004).

Research in early childhood education has established a connection between self-handicapping behaviors and achievement goals. For instance, Midgley and Urdan (2001) identified a significant positive relationship between personal performance-avoidance goals and self-handicapping behaviors among 7th graders, while no such association was observed for personal performance-approach goals. Leondari and Gonida (2007) compared different age groups and concluded that students begin to adopt academic self-handicapping strategies earlier in their academic careers. They found that while academic achievement remains a significant predictor of self-handicapping behavior in the upper elementary grades, in the process of shifting to high school, performance-avoidance goals become a stronger predictor of self-handicapping than achievement itself.

Research studies have also highlighted the relationship between personal achievement goals, classroom goal structures, and self-handicapping behaviors (Midgley & Urdan, 2001; Urdan et al., 1998). Studies have shown that students who focus on personal performance goals and perceive a strong emphasis on performance in the classroom structure are more inclined to exhibit self-handicapping behaviors. In contrast, students who focus on mastery goals and perceive a classroom environment that promotes mastery are less prone to engage in self-handicapping behaviors (Leondari & Gonida, 2007; Urdan et al., 1998). However, it is essential to mention that changing the classroom goal structure will not have the same impact on all students, as indicated by other studies (Urdan, 2004).

2.2.4 Attitude

Student attitudes are a critical factor influencing success and sustained interest in programming education. Research indicates that well-designed learning experiences and supportive instructional practices can significantly enhance these attitudes. Studies have demonstrated that effectively structured courses and activities can maintain or even improve positive student dispositions toward programming (Asad et al., 2016; de Vink et al., 2023). This underscores the importance of fostering engaging learning environments that stimulate both interest and motivation. Furthermore, the implementation of supportive instructional approaches has yielded positive results. These approaches include the utilization of block-based programming environments (Deniz & Korucu, 2023; Lambić et al., 2021; Totan & Korucu, 2023) and the integration of foundational theoretical knowledge with practical activities (Taşdöndüren & Korucu, 2022). Such methods can empower students to overcome challenges, develop a deeper comprehension of programming concepts, and ultimately cultivate a more positive perception of the subject matter. However, it is essential to consider the developmental stage of the students. Lambić et al. (2021) discovered that younger students (7-8 years old) using a challenging curriculum experienced a decline in positive attitudes compared to older students. This finding underscores the significance of tailoring the difficulty and complexity of learning activities to students' capabilities in order to maintain positive dispositions. In conclusion, these studies emphasize the significance of employing engaging and well-structured instructional approaches that provide students with the necessary support to develop a positive attitude toward programming. This positive attitude is paramount for promoting student success and fostering continued interest in the field (Love, 2023).

2.2.5 Cheating Behavior

Within the academic field, cheating refers to the act of presenting the work or ideas of another individual as one's own, typically for the purpose of attaining higher grades. The relationship between cheating behavior and goal theory has been extensively studied, providing insights into how students' achievement goals can influence their propensity to engage in dishonest practices.

Studies examining the relationship between cheating behavior and goal orientations have shown that certain achievement orientations can lead to cheating behavior. Performance goals have been found to lead to cheating more frequently than mastery goals (Meece et al., 2006; Senko et al., 2011). Additionally, in the context of programming education, several studies have investigated the effect of pair programming on reducing or inadvertently encouraging cheating behavior among students. Collaborative learning, particularly within the context of programming education, is a widely employed approach, especially for practical tasks. When pair programming first became widespread, there were expectations that it would help teachers prevent cheating behaviors. It was believed that, due to peer pressure, students would work more systematically on their projects and have someone to assist them, thus reducing the need to cheat (Williams & Upchurch, 2001). However, subsequent studies have shown that this method can be susceptible to misinterpretation by students, who may perceive it as an opportunity for dishonest behaviors such as cheating. This misinterpretation could stem from a misconception that cheating is synonymous with collaborative learning and information sharing (Barros et al., 2021; Williams, 1999). Cheating behavior has been examined in various studies, particularly concerning undergraduate and graduate students in computer science (Schulz et al., 2023). In their study involving undergraduate students, Hawi (2010) identified cheating as one of the causal attributions for programming achievement. Another study emphasized the presence of cheating behavior among students even during the initial stages of gamification implementation (Ibanez et al., 2014).

2.2.6 Research on the Impact of Motivational Factors on Students' Learning of Computer Programming

There is an expanding body of research investigating the multifaceted nature of achievement in computer science education, especially in the context of learning programming. These studies go beyond the cognitive aspects of programming and explore how factors like students' motivation, emotional experiences, learning behaviors, and the classroom environment all interact to influence achievement.

Research by Shell et al. (2013) contributed to the expanding body of research that explores factors beyond cognitive abilities influencing achievement in computer science education. Prior research has primarily focused on the cognitive or technical dimensions of learning programming, such as syntax and algorithms. However, this study investigated how motivational orientations, emotional experiences in the classroom, and self-regulation strategies were associated with course grades, knowledge retention, and ultimately, the long-term learning of computational thinking in introductory CS-1 courses. Their work established a clear link between mastery-oriented goals and positive academic outcomes. Additionally, the study supported the connection between positive emotions in the classroom and higher achievement. These findings underscored the importance of fostering an environment that encourages students to set deep learning goals, a notion further emphasized by Peteranetz (2021). To achieve this, a 3x2 goal orientation framework was utilized, and the study encompassed two separate investigations. They observed a concerning decline in all approach goals (learning, performance, and task) in upper-level CS courses. While performance-avoid goals showed significant decreases, which could be positive, there were no significant changes in task/work avoidance goals. Similarly, a study by Shell et al. (2016) provided evidence that while initial motivations are important, they do not always translate into long-term success. Although the study initially focused on understanding students' reasons for enrolling in the course and identifying those at risk from the outset, it ultimately highlighted the significance of the course itself in influencing motivation and how motivation

evolves throughout the course. On the other hand, Hazley et al., 2015 investigated the dynamic nature of goal orientation in post-secondary STEM courses. They observed shifts in goal orientation throughout a semester, with some changes (increased task-approach goals, decreased learning-avoidance goals) positively impacting achievement. However, the results showed that changes in performance-approach goals were not strongly influenced by classroom climate, although negative emotions were linked to a decrease in these goals. The findings of these studies emphasize the importance of a positive learning environment.

Patterns of achievement goal orientations in programming education have been assessed not only in face-to-face education but also in distance education. Polso et al. (2020) investigated student motivation in an open online introductory programming course by identifying five distinct achievement goal orientation profiles using a person-oriented approach that incorporates appearance, normative, and mastery goals. Results of the study indicated that learners with combined mastery and performance goals displayed slightly better outcomes compared to those with low goals. The study found no significant link between goal orientation profiles and overall course grades. Similar results were obtained in another study where task avoidance, self-approach/avoidance, and other-approach goals were not directly correlated with final exam scores (Tomić et al., 2020). This finding was in contrast to Shell et al.'s (2016) study, which identified goal orientation as a key factor for student success. While social comparison aspects of goal orientation, which focus on outperforming others, were not strongly linked to achievement in some studies (Tomić et al., 2020), others suggested potential beneficial outcomes. Peteranetz (2021) observed a decrease in performance-avoidance goals (fearing looking bad), which might be a positive development, as it could indicate a shift towards a more growth-oriented mindset, where challenges are seen as opportunities for learning. In another research, Gaddy and Ortega (2022) explored student enrollment decisions using a novel approach: virtual reality (VR). This innovative method revealed that scenarios highlighting goal orientation and career opportunities significantly influenced participants' enrollment decisions in CS courses, whereas focusing on

demographics had a negative impact. This suggests that potential CS students are more engaged by messages that connect to their aspirations and future goals rather than those that emphasize demographic characteristics.

In conclusion, student success in CS education transcends technical skills. A supportive learning environment that fosters goals and actively acknowledges the dynamic nature of motivation is crucial. By exploring the motivational factors, educators can develop targeted interventions to nurture student engagement, address challenges specific to gender, and empower students to navigate the evolving field of CS education.

2.3 Programming Education for Young Learners

There is a growing worldwide interest in the instruction of programming skills at elementary, middle, and high schools. In many countries, there has been intensive work carried out by governments on incorporating computer programming into school curricula. A wide range of studies explore the efficacy of programming languages that use blocks as their primary method of coding in teaching computer programming fundamentals. These studies highlight both the advantages and drawbacks of block-based programming tools. These tools affect students' motivation, interest, and engagement in a positive way compared to traditional methods. For example, a study by Ouahbi et al. (2015) explored the impact of block-based programming on high school students' motivation in programming. Researchers divided science majors into groups learning with either Scratch, a game-creation platform, or the traditional Pascal language. Students using Scratch displayed significantly higher interest in continuing programming compared to Pascal groups. Engagement with Scratch was also evident, as 85% of those students installed it on their home computers, far exceeding the 17.2% in the Pascal groups. Students were drawn to block-based programming due to its user-friendly nature. Compared to text-based programming, block-based environments offer visual cues, visual manipulation of code blocks, and natural language labels, making them easier

for novice learners to grasp. These environments also support learning by offloading memory tasks through block design (shape and color). Block-based programming, therefore, provides a valuable foundation for learning core programming concepts (Weintrop & Wilensky, 2015).

However, block-based programming might be perceived as less powerful than text-based programming. These tools may not offer the same level of complexity and power as traditional text-based programming languages (Weintrop & Wilensky, 2015). Block-based programming tools enhance the initial learning experience while preparing students for more complex programming tasks. Although block-based programming environments are highly effective for beginners and provide a smooth transition to text-based programming languages (Bau et al., 2017), they might be perceived as less powerful than text-based programming. These tools may not offer the same level of complexity and power as traditional text-based programming languages. Additionally, some research studies highlight the challenges associated with moving from block-based to text-based programming environments (Weintrop & Wilensky, 2015). To address this challenge, Bau et al. (2017) suggested a dual-mode approach that proposes bidirectional mode switching between block and text representations, leveraging the ease of blocks for learning syntax and the efficiency of text-based coding.

In the literature, there are numerous efforts to develop or adapt the computer programming self-efficacy scales designed to measure students' self-efficacy in programming across middle, high school, and undergraduate levels (Altun & Kasalak, 2018; Askar & Davenport, 2009; Cesur Özkara & Yanpar Yelken, 2020; Karalar, 2023; Kittur, 2020; Korkmaz & Altun, 2014; Kukul et al., 2017; Ramalingam & Wiedenbeck, 1998; Tsai et al., 2019). The systematic literature review conducted by Luxton-Reilly et al. (2018) mentioned that academic success is demonstrably influenced by students' self-efficacy and engagement. Research suggested that female and minority students tend to exhibit lower levels of self-efficacy compared to their peers.

A variety of pedagogical approaches have been investigated in computer education research. For instance, the master's thesis by Erdem (2018) investigated how 5th graders learned Scratch programming through two different approaches: traditional face-to-face instruction and flipped learning with technology support. The research revealed no substantial difference in educational achievements between the two teaching methods. Wells LeRoy's (2022) dissertation explored the potential of Minecraft for teaching logic gates with the participation of 122 college students. The study investigated two instructional design principles, guided discovery and pretraining, with a particular focus on their impact on cognitive load. While no notable disparities were observed in the learning outcomes between discovery approaches, Minecraft groups learned to build logic gates more effectively than the PowerPoint group. Besides, students in the direct instruction condition experienced significantly higher extraneous cognitive load.

While the importance of programming education has gained widespread recognition, the effective assessment of student learning in this domain has become a prominent area of research (Grover, 2020; Newton et al., 2021). Some researchers have explored a variety of assessment methods to assess the comprehension of programming principles among students. Some researchers attempted to develop traditional written exams. To this end, Grover, (2020) developed and evaluated a summative paper-hand assessment for measuring student learning in introductory programming courses tailored for grades 6-8 in middle school. This assessment incorporated a combination of multiple-choice and open-response question formats, all focusing on core programming concepts (“variables, expressions, loops, conditionals, and abstraction”) using Scratch as a familiar platform for students (p. 678). The analyses of validity, reliability, and item discrimination, coupled with the results of pre-and-post tests, suggest the assessment's effectiveness as a reliable measure of learning in introductory programming. Notably, this study also provides evidence for the effectiveness of well-designed multiple-choice items in assessing the comprehension of programming concepts by students. Another study by Newton

et al. (2021) demonstrated the effectiveness of the Evidence-Centered Design (ECD) framework in developing assessments for high school computer science courses.

The limited availability of computers in computer science classrooms has, to some extent, necessitated the adoption of pair programming. This circumstance may account for the significant amount of research devoted to exploring the impacts of pair programming. Albayrak and Polat (2022) carried out a mixed-methods study to investigate the experiences of students with pair programming. This study underscores the benefits of pair programming at the undergraduate level. Throughout a semester-long programming course, students worked in pairs, completed assessment forms after each lesson, and participated in in-depth interviews at the term's end. The findings revealed that students generally had positive experiences with pair programming, reporting enhanced academic performance, faster problem-solving, increased motivation, reduced anxiety, and improved communication skills. However, some challenges were noted, such as disagreements on problem-solving approaches and difficulty progressing when stuck together. Despite these challenges, the study suggested that pair programming could be a valuable teaching method, potentially addressing issues like student motivation and course completion rates.

2.3.1 Block-based Programming Environments

Block-based programming environments are commonly employed in early grades to instruct students lacking previous exposure to programming. Numerous block-based environments have been developed for teaching programming to young learners and novice programmers, including Scratch, Code.org, MIT AppInventor, Alice and CodeAcademy, among others. These tools enable students to create programs, games, applications, and animations without the need to type commands and deal with syntax errors, as is typical in traditional text-based programming languages. Block languages reduce the cognitive load for new programmers by eliminating syntax frustration (Bau et al., 2017; Luxton-Reilly et al., 2018). Moreover, these applications can be utilized on computers, laptops, or mobile phones. Through these

platforms, students can gain a comprehensive understanding of fundamental programming concepts like algorithms, loops, conditional statements, variables, functions, and events. These introductory programming environments leverage block-based programming languages, where each block encapsulates a specific programming concept. Learners construct executable computer programs by manipulating and connecting these blocks, similar to assembling a puzzle through a drag-and-drop interface. Block-based programming relies heavily on visual design to guide users. The shapes of the blocks themselves hint at their function, while colors categorize functionally similar blocks. Additionally, each block is clearly labeled, explicitly describing its purpose. One of the significant roles of visual cues is to facilitate understanding, while instructional scaffolding helps learners grasp fundamental programming concepts more effectively (Bau et al., 2017). Most block-based programming tools offer a dedicated workspace where users can visualize the execution of their program constructs. These tools then provide real-time visual or auditory feedback to the user, indicating the validity of the constructed program (Weintrop & Wilensky, 2015). Some block-based code editors, such as Blockly, offer the simultaneous display of the user's constructed code in both a visual block format and its corresponding text-based representation in specific programming languages.

Block-based programming environments serve as a novice-friendly introduction to programming fundamentals, acting as a stepping stone for a future transition to text-based programming languages. Despite their drag-and-drop interface, block-based tools maintain fidelity to core programming concepts. They incorporate instructional scaffolding similar to structured editors, ensuring learners' acquisition of essential programming principles while experiencing the core tenets of code writing. Essentially, block-based programming environments offer a transparent and accessible initial experience with programming, while establishing a foundation for further exploration within the realm of more complex programming languages that rely on text for coding (Weintrop & Wilensky, 2015).

Code.org

Numerous platforms are utilized at the K-12 level to teach programming, with Code.org being one of the most widely implemented in Turkey, particularly in middle schools, for introducing children to the fundamentals of computer programming. Code.org is an educational visual programming environment dedicated to broadening access to computer science education and ensuring its availability to all, with a particular emphasis on children and young learners. This initiative also conducts the annual Hour of Code campaign, engaging over 15% of students globally (Code.org, 2024). According to their 2022 Annual Report, it has amassed 80 million student accounts, with 47% identifying as female or gender-expansive and 48% representing underrepresented racial or ethnic groups (Code.org, 2022). By using this platform, children can design and develop their own games, animations, and applications. For educators and schools, Code.org provides resources, including lesson plans to help integrate computer science curricula into classrooms.

2.4 Summary

Computer programming, characterized by its cognitive complexity, imposes a significant cognitive load on learners, challenging their working memory. Cognitive Load Theory provides a valuable framework for addressing these challenges in programming education. The inherent complexity of programming, due to the integration of numerous new concepts, increases cognitive load and can hinder learning and problem-solving. Research has applied CLT to develop instructional strategies and tools aimed at optimizing cognitive load in programming education. Effective learning is achieved by minimizing extraneous load and optimizing intrinsic and germane loads. Innovative educational tools and techniques, informed by CLT, address the complexities of programming education.

Research on student motivation in computer science education reveals a complex interplay of factors influencing learning outcomes. While mastery goals are consistently associated with favorable academic outcomes, the role of performance goals is more differentiated. Performance-approach goals can be beneficial under certain conditions, but performance-avoidance goals are generally detrimental. The concept of classroom goal structures adds another layer, influencing their engagement and academic achievement. The influence of classroom goal structures, shaped by teacher practices and feedback, significantly impacts students' motivation and achievement. In conjunction, self-efficacy, a key determinant of motivation, is influenced by various factors, including past experiences, social support, and emotional states. Students' perceptions of their abilities in programming are shaped by their interactions with the subject matter and the learning environment.

Contextual factors such as gender, socioeconomic status, and learning styles also contribute to student motivation and achievement in computer science. The use of gamification and interactive learning environments can positively impact students' attitudes and motivation, but careful consideration must be given to the developmental levels of learners. Besides that, academic self-handicapping strategies are closely tied to performance-avoidance goals and are supported in environments that emphasize performance over mastery. Additionally, the occurrence of cheating behavior, which is notably observed in programming education, highlights the ethical concerns related to performance goals. Collaborative learning methods, while intended to enhance learning, can inadvertently facilitate dishonest behaviors if not carefully managed.

Block-based programming languages have been recognized as a successful tool for introducing fundamental programming concepts, offering a user-friendly approach that enhances student motivation and engagement compared to traditional text-based methods. Studies have shown that block-based platforms significantly increase students' interest and engagement and simplify code manipulation, making it accessible for novices. Despite their benefits, block-based environments are sometimes viewed as less powerful than text-based languages. Various pedagogical

strategies, including traditional instruction and innovative approaches like game-based learning, have been explored, with mixed results regarding their effectiveness in improving learning outcomes. Pair programming has been identified as a beneficial practice, enhancing academic performance and reducing anxiety, though it also presents challenges, such as conflicts over problem-solving approaches.

In conclusion, effective programming education requires a multifaceted approach that considers cognitive, motivational, and contextual factors. Cognitive Load Theory offers valuable insights into optimizing learning by minimizing cognitive overload and maximizing meaningful engagement with the material. Understanding and addressing students' motivation, including the interplay of mastery and performance goals, self-efficacy, attitude, and classroom goal structures, is crucial for fostering a positive learning environment. The strategic use of block-based programming environments can serve as an effective entry point to the field. Additionally, research on the impact of pedagogical approaches, such as pair programming, is essential for enhancing student learning outcomes. By addressing the cognitive, motivational, and contextual challenges of programming education, educators can lead to the development of more engaging and efficient learning experiences that empower students to succeed in this rapidly evolving field.

CHAPTER 3

METHODOLOGY

This chapter outlines the research methodology employed in this study. Initially, the research questions were identified. Following this, the chapter introduces the participants, details the research design, describes the study procedure, and elaborates on the data collection instruments. Furthermore, it addresses the pilot study, the implementation of the main study, the data analysis procedures, issues of validity and reliability, and ethical considerations.

3.1 Research Questions

To comprehensively explore the factors affecting the learning of basics of computer programming among middle school students, this study formulated the following questions:

1. Is there a significant difference in cognitive load experienced by students across seven fundamental programming topics?
2. Is there a significant difference in students' PALS (personal achievement goal orientations, perception of classroom goal structures, academic-related perceptions, beliefs and strategies), attitudes towards coding education, achievement in mathematics, achievement in reading comprehension, achievement in coding, and cognitive load scores based on their gender?
 - a. Is there a significant difference in students' PALS scores based on their gender?
 - b. Is there a significant difference in students' attitudes toward coding education scores based on gender?
 - c. Is there a significant difference in students' mathematics scores based on gender?

- d. Is there a significant difference in students' reading comprehension scores based on gender?
 - e. Is there a significant difference in students' coding achievement scores based on gender?
 - f. Is there a significant difference in students' cognitive load scores across seven fundamental programming topics based on gender?
3. Is there a significant difference in PALS (personal achievement goal orientations, perception of classroom goal structures, academic-related perceptions, beliefs and strategies), attitudes towards coding education, achievement in mathematics, achievement in reading comprehension, achievement in coding, and cognitive load scores between students from urban schools and suburban schools?
- a. Is there a significant difference in PALS scores between students from urban schools and suburban schools?
 - b. Is there a significant difference in attitudes toward coding education scores between students from urban schools and suburban schools?
 - c. Is there a significant difference in mathematics scores between students from urban schools and suburban schools?
 - d. Is there a significant difference in reading comprehension scores between students from urban schools and suburban schools?
 - e. Is there a significant difference in coding achievement scores between students from urban schools and suburban schools?
 - f. Is there a significant difference in cognitive load scores across seven fundamental programming topics between students from urban schools and suburban schools?
4. How do research variables predict students' achievement scores in programming?
5. What are the students' experiences and opinions on the factors that affect their learning fundamentals of programming?

3.2 Participants

3.2.1 Participants in the Quantitative Phase

The present study was conducted in three public middle schools situated within the Rize province. Participant selection employed a nonprobability convenience sampling method. While acknowledging limitations in generalizability due to the potential for self-selection bias, this method was chosen for its pragmatic advantages. Considering the research questions at hand, it was reasoned that a convenience sample drawn from these schools could provide appropriate information to test and investigate the research questions (Creswell, 2012). A total of 281 fifth-grade students were enrolled in the three participating schools. Of those students, 199 who regularly attended Information Technologies and Software (ITS) classes, consistently completed the data collection tools and obtained parental consent were selected as participants for the study on a voluntary basis.

Table 3.1 provides a distribution of participants from different geographical locations. School A, classified as an urban school, had the highest number of participants with 112 students, which constitutes 56.3% of the total sample. On the other hand, the other two schools are classified as a suburban school. Seventy-four participants of the study were from School B, accounting for 37.2% of the total participants. School C had the fewest participants among the schools, with 13 students making up 6.5% of the sample.

Table 3.1 Participants of the Study by Schools

Characteristics		<i>f</i>	%
Urban School	School A	112	56.3
	School B	74	37.2
Suburban School	School C	13	6.5

Table 3.2 provides an analysis of the characteristics of participants from urban and suburban schools. As seen in the table, 92 female and 107 male students participated in this study. A disparity was observed in the ownership of computers at home, with 80.4% of urban school students having a computer, compared to 55.2% of suburban school students. Household internet access was almost universal among urban students (99.1%) but slightly lower among suburban students (93.1%). The frequency of computer usage also varies, with urban students using computers more frequently on a weekly basis (38.4%) compared to suburban students (21.8%). Prior coding experience is more common among urban students (27.7%) than suburban students (11.5%). Overall, these findings emphasize the differences in access to technology and prior experience between students from urban and suburban schools.

Table 3.2 Characteristics of the Participants

Characteristic	Category	Urban		Suburban		Total	
		<i>f</i>	%	<i>f</i>	%	<i>f</i>	%
Gender	Female	53	47.3	39	44.8	92	46.2
	Male	59	52.7	48	55.2	107	53.8
Having a computer at home	Yes	90	80.4	48	55.2	138	69.3
	No	22	30.7	39	44.8	61	30.7
Household internet	Yes	111	99.1	81	93.1	192	96.5
	No	1	0.9	6	6.9	7	3.5
Frequency of computer usage	Never	14	12.5	23	26.4	37	18.6
	A few days a month	5	4.5	9	10.3	14	7.0
	A few days a week	43	38.4	19	21.8	62	31.2
	Less than 1 hour a day	15	13.4	12	13.8	27	13.6
	1-3 hours a day	25	23.3	15	17.2	40	20.1
	More than 3 hours a day	10	8.9	9	10.3	19	9.5
Prior coding experience	No prior experience	81	72.3	77	88.5	158	79.4
	With prior coding experience	31	27.7	10	11.5	41	20.6

The education levels of the parents of participants were provided in Table 3.3. Most participants' mothers and fathers had graduated from high school (45.2% of mothers and 44.2% of fathers). Similarly, a high percentage of urban mothers (47.3%) and suburban mothers (42.5%) had high school degrees. A significant portion of urban mothers hold bachelor's degrees (17.9%), whereas 6.9% of suburban mothers attained this level of education. Conversely, suburban mothers were more likely to have primary school degrees (23.0%) and middle school degrees (21.8%). A small percentage of mothers in urban areas had master's or PhD degrees (1.8%), while this level of education was not present among suburban mothers. Additionally, the percentage of illiterate mothers is higher in suburban areas (4.6%) compared to urban areas (0.9%).

Table 3.3 Parental Education Level of the Participants

Characteristic	Category	Urban		Suburban		Total	
		<i>f</i>	%	<i>f</i>	%	<i>f</i>	%
The education level of the mother	Illiterate	1	0.9	4	4.6	5	2.5
	Primary school degree	15	13.4	20	23.0	35	17.6
	Middle school degree	11	9.8	19	21.8	30	15.1
	High school degree	53	47.3	37	42.5	90	45.2
	Associate degree	10	8.9	1	1.1	11	5.5
	Bachelor's degree	20	17.9	6	6.9	26	13.1
	Master's/PhD degree	2	1.8	-	-	2	1.0
The education level of the father	Illiterate	1	0.9	2	2.3	3	1.5
	Primary school degree	8	7.1	15	17.2	23	11.6
	Middle school degree	17	15.2	16	18.4	33	16.6
	High school degree	50	44.6	38	43.7	88	44.2
	Associate degree	8	7.1	7	8.0	15	7.5
	Bachelor's degree	25	22.3	9	10.3	34	17.1
	Master's/PhD degree	3	2.7	-	-	3	1.5

When the education level of fathers was examined, similar trends were observed. A larger proportion of urban fathers (44.6%) and suburban fathers (43.7%) held high school degrees. Bachelor's degrees were more common among urban fathers (22.3%) than suburban fathers (10.3%). However, suburban fathers had higher percentages of primary school degrees (17.2%) and middle school degrees (18.4%) compared to urban fathers. A small number of urban fathers held master's or PhD degrees (2.7%), whereas this level of education was not present among suburban fathers. The percentage of illiterate fathers is slightly higher in suburban areas (2.3%) compared to urban areas (0.9%).

3.2.2 Participants in the Qualitative Phase

At the end of the implementation phase, semi-structured interviews were carried out with selected students. Participants were purposively selected based on teacher recommendations to represent a range of academic achievements in the ITS course. Three students were selected from each of the ten participating classes (six from School A, three from School B, and one from School C), one representing low, one moderate, and one high academic achievement. As detailed in Table 3.4, the interview participants included 14 female and 16 male students.

Table 3.4 Demographic and School Information of Interviewed Students

ID	Gender	School	Geographical School Location	Class
S1	Male	School A	Urban	5A
S2	Female	School A	Urban	5A
S3	Male	School A	Urban	5F
S4	Female	School A	Urban	5F
S5	Male	School A	Urban	5F
S6	Male	School A	Urban	5D
S7	Male	School A	Urban	5D
S8	Female	School A	Urban	5D
S9	Female	School B	Suburban	5A
S10	Male	School B	Suburban	5A
S11	Male	School B	Suburban	5A
S12	Male	School B	Suburban	5C
S13	Female	School B	Suburban	5C
S14	Female	School B	Suburban	5C
S15	Male	School B	Suburban	5B
S16	Female	School B	Suburban	5B
S17	Male	School B	Suburban	5B
S18	Male	School A	Urban	5B
S19	Male	School A	Urban	5B
S20	Female	School A	Urban	5B
S21	Female	School A	Urban	5E
S22	Female	School A	Urban	5E
S23	Male	School A	Urban	5E
S24	Male	School A	Urban	5C
S25	Female	School A	Urban	5C
S26	Male	School A	Urban	5C
S27	Female	School C	Suburban	5A
S28	Female	School C	Suburban	5A
S29	Male	School C	Suburban	5A
S30	Female	School A	Urban	5A

3.3 Research Design of the Study

The current study aimed to explore and analyze the factors that influence middle school students' acquisition of foundational computer programming knowledge and skills. To achieve this objective, the study utilized a mixed-methods research design that offers a comprehensive approach to exploring complex research questions.

A mixed-methods research design outlines a systematic approach for integrating quantitative and qualitative data within the same study. This approach strategically leverages the advantages of both qualitative and quantitative methods. Quantitative methods are useful in educational research for identifying patterns and relationships through numerical data analysis (Creswell, 2015). Quantitative methods play a crucial role in educational research by facilitating the identification of patterns and relationships through numerical data analysis. On the other hand, qualitative methods offer distinct advantages over quantitative approaches in several ways: they investigate participants' inner experiences, explore how individuals construct and understand meaning in their world, provide in-depth exploration in emerging research areas, identify variables for further investigation through quantitative methods, and foster a holistic and comprehensive understanding of phenomena (Corbin & Strauss, 2012). Combining these methodologies can yield a more comprehensive picture of the phenomenon under study than either method could achieve alone.

Although mixed-methods studies have drawbacks, such as the need for substantial time, resources, and the researcher's expertise in both qualitative and quantitative research methods, they offer a multitude of advantages for research. One strength of this method is its capacity to enrich the understanding of underlying relationships between variables. Furthermore, mixed-methods research facilitates an in-depth exploration of the relationships between variables. Additionally, mixed-methods designs can contribute to the confirmation or cross-validation of relationships identified between variables (Fraenkel et al., 2012).

In mixed-methods design, the combination of qualitative and quantitative data goes beyond simply aggregating them into a single dataset. Instead, this process involves a rigorous approach that aims to achieve a comprehensive and multifaceted understanding of the research phenomenon (Creswell, 2015). Within the field of mixed-methods research, various frameworks exist for conceptualizing research designs. For instance, Creswell (2015) categorized mixed-methods designs into basic and advanced groups. While this framework provides a general structure, Fraenkel

et al. (2012) offered a complementary perspective by identifying three specific designs frequently used in educational research: exploratory, explanatory, and triangulation designs. According to the framework defined by Creswell (2012, 2015), basic designs, encompassing convergent, explanatory sequential, exploratory sequential, and embedded designs, involve the collection and analysis of quantitative and qualitative data in different ways. In a convergent parallel design, data from both methodologies are collected simultaneously and analyzed independently, with subsequent comparison to identify convergence or divergence in the findings. Sequential designs involve data collection in two distinct phases. The explanatory sequential design begins with the collection of quantitative data, which is then followed by qualitative data to explain the “why” behind the quantitative findings. The exploratory sequential design follows the opposite sequence, starting with qualitative data for the initial exploration of the variables associated with the phenomenon and then utilizing quantitative data to explore the relationships between these variables (Fraenkel et al., 2012). The embedded design is similar to the convergent and sequential designs in that quantitative and qualitative data are collected concurrently or sequentially. However, in this type of design, one data type serves a supplementary role in enhancing the understanding derived from the primary data type. While these basic mixed-methods designs are identified as distinct approaches, they can be nested within advanced designs. Examples include framing a basic design within an experiment, a social justice inquiry, or an evaluation process (Creswell, 2015).

Mixed-methods designs are helpful for a comprehensive understanding of complex educational phenomena by integrating qualitative and quantitative approaches. This study aims to investigate the multifaced nature of learning programming that involves not only cognitive aspects but also social and behavioral aspects. In this study, the quantitative data identified predictors of programming achievement. The qualitative component then explored deeper into these relationships, providing richer insights and different perspectives. By integrating the findings from both approaches, the study aimed to achieve a more thorough comprehension of the ways

in which different factors impact students' learning of programming. In particular, in this study, a convergent embedded mixed-methods design, in which a qualitative component is embedded within a quantitative design, was utilized. Both quantitative and qualitative data were collected simultaneously in order to explore different research questions, where the focus is on the quantitative data, and the qualitative data supports the quantitative data (Fraenkel et al., 2012). The qualitative data served to enrich our understanding of the quantitative findings by providing insights into the underlying reasons or experiences associated with the quantitative results.

3.4 Procedure of the Study

3.4.1 Preliminary Investigation

The preliminary investigation served as a critical first step in preparing for this study. It focused on the phenomenon of "teaching programming to children" within the context of Turkey's middle school curriculum. At the time the study commenced, the teaching of programming to younger age groups was just beginning to become widespread in Turkey, even though the relevant learning outcomes had been included in the curriculum previously. This study aims to evaluate factors that affect students' programming learning within the existing teaching processes rather than intervening in the learning environment.

When the research concept was initially developed, the "Middle School and Imam Hatip Middle School Information Technologies and Software Course (Grades 5, 6, 7, and 8) Curriculum," published by the National Ministry of Education, Board of Education and Training in 2012, was in effect (since access to past curriculum documents is only possible through an official application/petition to the Ministry of National Education, this curriculum cannot be referenced, Presidency of the Board of Education, n.d.). This standard-based curriculum comprises four categories of competencies along with standards that express the knowledge and skills pertaining to information and communication technologies for each competency. In the

curriculum, there are no specific levels or topics designated for teaching a particular grade. Instead, the selection of levels and current topics is left to the discretion of the teacher. Subsequently, in 2018, a new curriculum for grades five and six was published by the Ministry of National Education along with the Teacher's Guide and Student Materials.

Therefore, this investigation was conducted to gain an understanding of the current state of programming instruction in Turkey's middle schools, determine the research needs, and tailor the research design accordingly. This involved conducting exploration with IT teachers who had direct experience teaching programming to children. In this regard, data were collected from 319 volunteer IT teachers across 71 Turkish cities. A total of 303 teachers completed the online survey, while 16 participated in interviews. Data was gathered through online surveys and semi-structured interviews. The survey consisted of eight general demographic questions, two yes-no questions, eight multiple-choice questions with closed-ended response options, and thirty-two open-ended questions. In the survey, close-ended questions were followed by open-ended questions, where participants described their experiences in detail based on their responses to the closed-ended questions. Consequently, the number of open-ended questions varied for each participant. The survey questions were reviewed by two subject matter experts and checked by a Turkish language expert for any obscure expressions. Additionally, a pilot survey was conducted with two IT teachers. The semi-structured interview questions were developed using the survey questions as a guide.

LimeSurvey, an open-source online survey application, was used to develop and administer the survey. The survey was published on a personal website. The survey invitation, either as a text or image, was disseminated on various social media platforms to invite IT teachers to participate in the study. Furthermore, an invitation letter was emailed to the corporate mail addresses of schools. Following the survey administration, interviews were conducted with volunteer teachers. At the outset of each interview, participants were briefed on the study's purpose and permission was obtained for audio recording. The interviews were carried out over the telephone,

and each session was recorded. These interviews lasted approximately 20 to 55 minutes. The data obtained from this preliminary investigation and the main study were analyzed using the same qualitative data analysis procedure. The detailed analysis procedure is presented under the 'Qualitative Data Analysis' section (p. 78).

The following section presents the main results obtained from the examination of the survey and interviews, along with a discussion of how these findings shaped the subsequent stages of the research study:

- Survey results showed that 63.04% ($N = 191$) of the teachers involved in the research incorporated programming instruction in their classes. The primary reason cited by teachers who did not incorporate programming instruction was technological deficiencies.
- According to the interview results, students entering the Information Technologies and Software course in grades 5 or 6 have little to no prior exposure to foundational information technology concepts.
- While survey participants indicated which learning objectives they included in their lessons, interviews with teachers revealed confusion about integrating these objectives into their lesson plans. Some teachers were unaware of the new curriculum. Additionally, some teachers did not follow a specific annual or daily plan and taught the Information Technologies and Software courses independently of the curriculum's learning objectives. Even when using common plans provided by their departments or shared on online platforms, some teachers stated that they focused on solving specific puzzles, particularly those using the Code.org coding environment, instead of ensuring alignment with specific learning objectives. On these platforms, teachers selected examples or courses for their students to complete each week but often overlooked aligning them with specific learning objectives.

Considering the preliminary research findings, the following adjustments were made to the study:

- The absence of standardized programming instruction practices in middle education, such as inconsistencies in curriculum implementation and variations in the specific programming outcomes addressed, posed a significant challenge for conducting a comprehensive study that encompasses data from multiple schools. This lack of standardization could lead to inconsistencies in the pace and depth of programming instruction, making it difficult to draw meaningful comparisons and identify patterns across different learning environments. To address this challenge and ensure the collection of consistent data that facilitates meaningful analysis, the establishment of weekly learning objectives and lesson plans was deemed essential. Weekly learning objectives were intended to serve as a common framework for all participating schools, ensuring that students are exposed to a consistent sequence of programming concepts and skills throughout the study period. Additionally, aligned lesson plans were intended to provide teachers with a detailed guide for each week's instruction, including activities and resources.
- Given the prevalence of Code.org as the preferred block-based coding platform for introductory programming instruction at the middle level and recognizing the teachers' existing familiarity with this tool, the decision was made to adapt and utilize lesson plans from Code.org curriculums. For this purpose, to ensure appropriate difficulty and alignment with middle-level learning objectives, Course F, originally designed for fifth grade, was carefully reviewed and adapted.

3.4.2 Adaption Process of the Lesson Plans

In preparation for the study, 27 of the 28 programming-related lesson plans from Course F on Code.org were translated from English to Turkish. This translation process involved two language experts: one for the initial translation and another for a thorough review. A researcher then made the final corrections to ensure accuracy and clarity. A pilot study was conducted to evaluate the feasibility of the lesson plans in a classroom setting. The lesson plans were distributed to 15 IT teachers based on their students' readiness and pre-learning levels, as well as teacher preferences. The teachers were informed about the research goals and participated voluntarily. They were encouraged to contact the researcher with any questions throughout the pilot. The researcher provided support via phone calls, text messages, or in-person visits to the schools. After implementing each lesson plan, teachers were asked to complete a Lesson Plan Evaluation Form (Appendix A) and send it electronically to the researcher. Data from the Lesson Plan Evaluation Form was used to identify suggestions and problems reported by the teachers, as listed below:

- It was reported that some lesson plans, especially those with extensive unplugged activities, could not be completed within a two-hour class period.
- Providing the materials to be used in the course for unplugged activities was not easy for some teachers. The preparation process for these courses was considered time-consuming by some of them.
- The pilot study revealed issues with clarity in some lesson plans, especially those with extensive unplugged activities. Teachers found the language confusing and the instructions insufficient, making it difficult to understand the intended activities. To address this, simplifying the language and providing more detailed explanations were suggested.
- Teachers recommended incorporating more in-class practice with digital puzzles before transitioning to independent or paired work. This would provide scaffolding to ensure student understanding.

- It was stressed that sometimes students had difficulty making connections between activities and the related concepts covered in the lesson.
- Concepts requiring specific mathematical knowledge, such as angles, presented challenges for student comprehension and application.
- Feedback from the pilot study highlighted that some lesson plan elements (e.g., playing cards) were considered distracting by the participating teachers.

The pilot study yielded feedback from teachers on the lesson plans, highlighting areas for improvement. Based on these findings lesson plans were revised.

3.4.3 Lesson Plan Evaluation Workshop

Following the implementation and revision of the lesson plans based on teacher feedback, a two-day workshop was conducted with IT teachers. The workshop aimed to refine the piloted course content for the research study, focusing on teaching programming fundamentals to novice students in fifth grade. The workshop began with participants collaboratively identifying suitable learning objectives from the official 2018 curriculum. These objectives then guided the selection of lesson plans and activities. The teachers structured a ten-week program by selecting appropriate elements from both adapted Code.org lesson plans and the official 5th Grade Computer Technologies and Software Teacher Guide. The workshop involved four IT teachers, with two participants working in public schools and the other two employed by private schools. Lesson plans were distributed to the teachers in advance of the workshop. They were requested to review the materials beforehand to facilitate a productive discussion during the sessions. Additionally, printed copies of the lesson plans were provided to each participant at the workshop's start for easy reference. To capture the workshop discussions and activities, all sessions were video recorded. This resulted in approximately 10 hours of data. Based on the researcher's field notes and the video recordings of the workshop sessions, the

targeted learning outcomes, corresponding lesson plans, and activities for the implementation of the study were identified.

3.5 Data Collection Instruments

3.5.1 Coding Achievement Test

The Information Technologies and Software course Coding Achievement Test was developed for 5th-grade students to evaluate their understanding of the basics of programming (see Appendix B).

3.5.1.1 Development of the Coding Achievement Test

Some items of the test were developed by the researcher through a literature review, while others were developed by revising the questions of the coding achievement tests developed in the previous research studies and the questions from the coding textbooks recommended by the interviewed IT teachers during the lesson plan development process. In the development of the questions regarding measuring competencies in block-based coding, code.org and Scratch block-based coding platforms were used. The candidate achievement test was developed with 46 items, which were formed based on the learning outcomes defined in the fifth-grade curriculum of the Information Technologies and Software course published by the Ministry of National Education in 2018.

As a first step towards evaluation of the achievement test, it was reviewed by an assessment and evaluation expert in terms of construct validation and reviewed by a language expert and an IT teacher for language suitability. Subsequently, as the second step, the Content Evaluation Panel was established, comprising four subject matter experts and nine IT educators. Each IT educator was employed at a public school and possessed over five years of experience in teaching programming at the middle school level, particularly in fifth and sixth grades. One of the subject matter

expert panelists was selected from the Department of Computer Programming, having graduated from the Department of Computer Education and Instructional Technology (CEIT). Another subject matter expert was chosen from the Department of Computer Engineering, also a graduate of CEIT. Besides that, one subject matter expert from the Department of Computer Programming with over ten years of experience in teaching programming at a vocational school and one subject matter expert from the Department of CEIT were included. The expert evaluation form, consisting of four questions, was subsequently developed to investigate the content validity of the instrument. It aimed to evaluate the appropriateness of the instrument for the target audience, as well as the comprehensibility and difficulty levels of the items.

Content validity was investigated with the question (1) "Does the item represent the property to be measured?" The response options for this question were: "Essential", "Useful but insufficient" and "Not necessary". Response options for the other questions ((2) Is the item appropriate for the target audience? (this question just asked panelists who were working at middle school and/or graduated from CEIT), (3) Is the item sufficiently clear?, (4) What is the difficulty level of the item?) were: "Appropriate", "Appropriate but needs revision" and "Not appropriate" for the second question, "Clear", "Clear but needs revision" and "Not clear" for the third question; and "Simple", "Medium" and "Difficult" for the last question (Yeşilyurt & Çapraz, 2018). Besides, a column labeled "comments" was added to the far right of the table to provide space for respondents to optionally add their comments regarding each item. At the end of the evaluation form, subject matter experts were also asked if they had any further comments regarding the overall test.

Forty-six candidate items were submitted to the panel for expert opinion. Panelists were asked to grade each item for each question on the evaluation form by selecting one of the given options. The content validity of the achievement test was evaluated by the determination of content validity rates by using the Lawshe technique (Lawshe, 1975). It was ascertained how many panelists selected the 'Essential' option

for each item, and then the content validity ratio (CVR) for each item was calculated by utilizing the following equation:

$$CVR = \frac{n_{e-N/2}}{N/2}$$

(n_e= Number of panelists indicating "essential", N= Total number of panelists)

The content validity index of individual items (I-CVI) was calculated by dividing the number of panelists considering an item as ‘essential’ by the total number of experts. CVR and I-CVR values for each item are presented in Table 3.5.

Table 3.5 Content Validity Values of the Test Items

Item	CVR	I-CVI	A (%)	C (%)	DL (%)		
					E	M	D
1	1.00	1.00	1.00	1.00	0.77	0.23	-
2	0.85	0.92	1.00	0.92	0.69	0.31	-
3	1.00	1.00	0.92	0.92	0.08	0.46	0.46
4	0.69	0.85	0.75	0.92	0.54	0.31	0.15
5	0.85	0.92	1.00	1.00	0.77	0.23	-
6	1.00	1.00	1.00	1.00	0.62	0.38	-
7	1.00	1.00	0.92	0.85	0.69	0.31	-
8	1.00	1.00	1.00	1.00	0.23	0.23	0.54
9	0.69	0.85	0.92	1.00	0.08	0.54	0.38
10	0.54	0.77	0.67	0.92	0.31	0.38	0.31
11	1.00	1.00	1.00	1.00	0.08	0.62	0.31
12	1.00	1.00	0.92	0.85	0.23	0.62	0.15
13	1.00	1.00	1.00	0.85	0.08	0.77	0.15
14	0.85	0.92	0.92	1.00	0.31	0.38	0.31
15	0.85	0.92	0.92	0.02	0.08	0.23	0.69
16	0.54	0.77	0.67	0.85	-	0.23	0.77
17	0.85	0.92	0.92	1.00	0.08	0.62	0.31
18	0.85	0.92	0.75	0.85	0.92	-	0.08
19	0.85	0.92	0.83	1.00	0.77	0.15	0.08
20	0.85	0.92	1.00	1.00	0.54	0.46	-
21	0.54	0.77	0.75	0.77	0.23	0.54	0.23
22	0.69	0.85	0.83	0.85	-	0.38	0.62

Table 3.5 Content Validity Values of the Test Items (cont'd)

23	0.54	0.77	0.83	0.85	0.15	0.46	0.38
24	0.54	0.77	0.75	0.77	0.08	0.62	0.31
25	0.54	0.77	0.83	0.85	0.15	0.62	0.23
26	0.85	0.92	0.75	0.46	0.69	0.15	0.15
27	1.00	1.00	0.83	0.92	0.54	0.46	-
28	0.08	0.54	0.58	0.62	0.31	0.46	0.15
29	1.00	1.00	1.00	0.85	-	0.85	0.15
30	1.00	1.00	0.92	1.00	0.08	0.62	0.31
31	0.85	0.92	0.92	0.92	0.77	0.15	-
32	0.69	0.85	0.92	1.00	0.92	0.08	-
33	0.69	0.85	0.92	0.92	0.69	0.15	0.15
34	0.54	0.77	0.92	0.85	0.46	0.38	0.15
35	0.54	0.77	0.75	0.77	0.38	0.38	0.15
36	0.23	0.62	0.75	0.77	0.31	0.54	0.15
37	0.85	0.92	0.83	0.92	-	0.69	0.31
38	1.00	1.00	1.00	0.92	0.15	0.69	0.15
39	0.85	0.92	0.92	0.92	0.08	0.46	0.46
40	1.00	1.00	1.00	0.92	0.23	0.54	0.23
41	1.00	1.00	1.00	0.77	0.38	0.62	-
42	1.00	1.00	1.00	0.77	0.08	0.15	0.77
43	0.85	0.92	0.83	0.85	0.23	0.69	0.08
44	0.85	0.92	0.92	0.85	-	0.54	0.46
45	0.69	0.85	1.00	0.92	0.38	0.38	0.23
46	1.00	1.00	1.00	1.00	0.08	0.54	0.38

CVI= 0.89, S-CVI/Ave=.94

Note. **A**= Appropriate, **C**= Clear, **DL**= Difficulty level of an item, **E**= Easy, **M**= Medium, **H**= Hard

Items were eliminated due to the critical CVR and I-CVI values ($\alpha = .05$), which were defined according to the panelist numbers. When the panel was composed of 13 panelists, a minimum CVR value of .54 (Ayre & Scally, 2014; Lawshe, 1975) and a minimum I-CVI value of .78 were required for any item to be valid and included in the instrument. Two items with a CVR value below .54 and eight items that achieved an I-CVI value below .78 were excluded from the test. Additional changes were also made to the items based on the comments and suggestions written by the panelists. Consequently, 36 items were identified to be included in the last

draft form of the achievement test. Following that, the content validity index (CVI) was determined for the entire test by computing the mean of CVR values of the 36 items that were kept, resulting in a CVI of .89 (Lawshe, 1975); and S-CVI/Ave value obtained by computing the mean of I-CVI values of all retained items as .94 which indicates the high content validity for the achievement test. The final draft form of the coding achievement test was piloted by administering it to the three fifth-grade students to ensure that all of the items were clear and understandable.

3.5.1.2 Item Analysis of the Coding Achievement Test

The draft coding achievement test was administered to 414 5th-grade students from public middle schools in Rize for the item analysis of the test. Each student's correct responses were encoded as 1, while the incorrect ones were encoded as 0. To calculate the discrimination levels of the items, the students' total scores were ranked from the highest to the lowest using SPSS statistical software, and the upper group and the lower group were identified using the critical value of 27 percent. Item discrimination index (D) was computed for each item by using the following equation $D = (UG - LG) / n$, where UG is the total number of students in the upper 27% ($n = 112$), and LG is the total number of students in the lower 27% ($n = 112$) who responded the item correctly. In addition, the $DL = (UG + LG) / n + n$ formula was used to find the difficulty levels of the items. The results of the analysis are displayed in Table 3.6. The test results showed that one question (Q4) was too difficult, four questions (Q3, Q12, Q20, and Q21) were easy, twelve questions (Q5, Q10, Q11, Q17, Q23, Q24, Q25, Q28, Q30, Q34, Q35 and Q36) were difficult and the remaining nineteen questions were moderately difficult. As seen in Table 3.6, the discrimination indices of the 23 items were ideal. Five items with discrimination indices within the normal range were deemed to be acceptable. However, eight items (Q4, Q5, Q11, Q17, Q24, Q25, Q30, and Q34) that had discrimination indices below .30, indicating too low or low discrimination power, were removed from the test.

Table 3.6 Item Analysis Results of the Coding Achievement Test

Item	DI	DI interpretation	DL	DI interpretation
Q1	0.54	Ideal	0.55	Moderately difficult
Q2	0.54	Ideal	0.58	Moderately difficult
Q3	0.36	Normal	0.77	Easy
Q4	-0.16	Unsatisfactory	0.19	Too Difficult
Q5	0.27	Low	0.35	Difficult
Q6	0.61	Ideal	0.50	Moderately difficult
Q7	0.51	Ideal	0.58	Moderately difficult
Q8	0.56	Ideal	0.56	Moderately difficult
Q9	0.41	Ideal	0.44	Moderately difficult
Q10	0.44	Ideal	0.37	Difficult
Q11	0.21	Low	0.30	Difficult
Q12	0.52	Ideal	0.72	Easy
Q13	0.62	Ideal	0.57	Moderately difficult
Q14	0.55	Ideal	0.58	Moderately difficult
Q15	0.56	Ideal	0.42	Moderately difficult
Q16	0.51	Ideal	0.58	Moderately difficult
Q17	0.19	Too low	0.37	Difficult
Q18	0.55	Ideal	.53	Moderately difficult
Q19	0.61	Ideal	.57	Moderately difficult
Q20	0.54	Ideal	.60	Easy
Q21	0.64	Ideal	.63	Easy
Q22	0.62	Ideal	.42	Moderately difficult
Q23	0.34	Normal	.35	Difficult
Q24	0.26	Low	.30	Difficult
Q25	0.13	Too low	.26	Difficult
Q26	0.54	Ideal	.54	Moderately difficult
Q27	0.38	Normal	.42	Moderately difficult
Q28	0.46	Ideal	.33	Difficult
Q29	0.35	Normal	.47	Moderately difficult
Q30	0.27	Low	.35	Difficult
Q31	0.71	Ideal	.43	Moderately difficult
Q32	0.47	Ideal	.54	Moderately difficult
Q33	0.51	Ideal	.42	Moderately difficult
Q34	0.24	Low	.34	Difficult
Q35	0.31	Normal	.30	Difficult
Q36	0.43	Ideal	.37	Difficult

Considering the findings from the analysis of the items, the final form of the fifth-grade coding achievement test was composed of 28 items. Cronbach's Alpha value was computed for retained 28 items as 0.84 indicating good internal consistency. Items and corresponding learning objectives are outlined in Table 3.7.

Table 3.7 The Distribution of the Items According to Learning Objectives

Learning Outcomes (Students will be able to...)	Item
IT.5.5.1.6. Explain the variables, constants and operations required to solve the problem.	1, 2
IT.5.5.1.7. Give examples of operators that can be used in problem solving.	3
IT.5.5.1.10. Use operators to solve a given problem.	10
IT.5.5.1.12. Explain the concept of algorithm.	6, 7
IT.5.5.1.13. Develop an algorithm for solving a problem.	13, 15
IT.5.5.1.14. Explain flowchart components and functions.	16
IT.5.5.1.15. Draw a flowchart for an algorithm.	20
IT.5.5.1.16. Debug an algorithm by testing it.	14, 21
IT.5.5.2.1. Explain the basic concepts of programming.	8, 9
IT.5.5.2.2. Recognize the interface and features of the block-based programming tool.	12, 18
IT.5.5.2.3. Create the appropriate algorithm to achieve the goals presented in the block-based programming environment.	22, 26, 28, 29, 31, 32, 33
IT.5.5.2.4. Explain the structure of linear logic.	-
IT.5.5.2.5. Develop algorithms using linear logic structure.	22
IT.5.5.2.6. Explain the decision structure and its functions.	23
IT.5.5.2.7. Develop algorithms with decision structures.	26, 31, 32
IT.5.5.2.8. Explain the loop structure and its functions.	19
IT.5.5.2.9. Create algorithms with loop structure.	28, 29, 33
IT.5.5.2.10. Debug the algorithms created for different structures by predicting the results of it.	27, 35, 36

3.5.2 Cognitive Load Scale

3.5.2.1 Translation and Adaption of the Scale

Cognitive Load Scale (CLS), an 11-point Likert Type scale developed by (Leppink et al., 2013) to assess three types of cognitive load (intrinsic, extraneous, and germane load) in statistic lectures, was adapted and its applicability for middle school students and programming teaching was verified. As seen in Table 3.8, CLS consisted of 10 items; three items measuring intrinsic load (items 1, 2, and 3), three items measuring extraneous load (items 4, 5, and 6), and four items (items 7, 8, 9, and 10) measuring germane load.

Table 3.8 Items of the Cognitive Load Scale

Items
1 The topic/topics covered in the activity was/were very complex.
2 The activity covered formulas that I perceived as very complex.
3 The activity covered concepts and definitions that I perceived as very complex.
4 The instructions and/or explanations during the activity were very unclear.
5 The instructions and/or explanations were, in terms of learning, very ineffective.
6 The instructions and/or explanations were full of unclear language.
7 The activity really enhanced my understanding of the topic(s) covered.
8 The activity really enhanced my knowledge and understanding of statistics.
9 The activity really enhanced my understanding of the formulas covered.
10 The activity really enhanced my understanding of concepts and definitions.

3.5.2.1.1 Translation process

For this purpose, firstly, ten items of the scale were translated into Turkish by three English language experts, and then these three translations were compared and combined into one common version by one Turkish language expert to ensure the naturalness of the language (Erten, 2012). Secondly, three different English language experts, blind to the original instrument, performed a back translation of these Turkish versions of the items into the original language (Geisinger, 1994).

CLS was adapted for the domain of Computer Science in different research studies (Harms et al., 2016; Morrison et al., 2014) and for the middle school context (Weng et al., 2018). In a similar way, items of the scale were adjusted to suit better to the terminology used in computer programming at the middle school level in this study. For this purpose, the terms used in 3 items were changed. Namely, the term “statistics” was replaced by the term “programming” as deemed appropriate by Leppink et al. (2013). Besides, the word “formulas” in items 2 and 9 was replaced by “algorithms/problems.” In addition to that, the 10-point Likert response was changed to a 5-point Likert format ranging from strongly to disagree (1) to strongly agree (5) in order to ensure middle school students understanding.

The items of the original English and the translated Turkish versions were compared and rated independently and blindly by three experts to check the linguistic and semantic equivalence of the two versions. The rating was realized on a 10-point Likert scale format, ranging from “not related at all” (1) to “100% synonymous” (10). The mean score of the raters’ responses ($M = 9.50$) showed that the Turkish version had a high level of equivalence with the original English version. In other respects, the items in both their original English and back-translated English versions were evaluated by another three English language experts for semantic equivalence again on a 10-point Likert scale format, ranging from “not related at all” (1) to “100% synonymous” (10). Results indicated a high level of equivalence between the back-translated and original English versions ($M = 9.27$). The opinions of the two Information Technologies and Software teachers were obtained for the clarity of the scale and the appropriateness of the translation to the concepts of programming. Scale was piloted with ten sixth-grade students (three students with low academic achievement, three students with average academic achievement, and four students with high academic achievement). In the selection of students, academic success in Information Technology and Software courses was taken into consideration with the guidance of the IT teacher. Students were asked to complete one activity (namely, Robot Route is Flow Chart) from the Second-term Materials Book published by the Ministry of Education in 2018 and then fill in the Cognitive Load scale. Then, they

filled in the Cognitive Load scale and three open-ended questions which aimed to determine the unclear statements for the students. Informal interviews were conducted with the students who had difficulties in understanding or answering the items on the scale. In accordance with the feedback received from the students, statements were revised by one IT teacher and 3 Turkish teachers working at a middle school by focusing on the statements that were not understood by the students. Herewith, the process of translation and adaption of 10 scale items was completed (see Appendix C).

3.5.2.1.2 Exploratory and Confirmatory Factor Analysis Results of the Cognitive Load Scale

The scale was applied to 804 sixth-grade students at the end of the two-hour Information Technologies and Software class at eight different middle schools in Rize. The data were gathered from students in the sixth grade as programming concepts were generally introduced to the students in the last one or two weeks of the fall semester for fifth-grade students in schools located Rize. As shown in Table 3.9, a total of 47 cases, including missing data, were excluded from the analysis, so 757 responses were analyzed.

Table 3.9 Distribution of Participants to Schools

School Name	N
School 1	29
School 2	169
School 3	55
School 4	189
School 5	69
School 6	70
School 7	52
School 8	124
Total	757

Exploratory Factor Analysis (EFA) was conducted to determine if the questions adapted from the original scale load onto three types of cognitive load. Additionally, Confirmatory Factor Analysis (CFA) was conducted to test the scale for measurement of specific cognitive load factors. Before conducting the analyses, the data were screened to identify univariate and multivariate outliers, as well as multicollinearity and violations of normality. When the original mean score of each item and the trimmed mean scores were compared, the results indicated that the extreme scores did not have a strong influence on the mean. Skewness and kurtosis values of each item were all within the cutoff point value ± 2 for large samples (Tabachnick & Fidell, 2012), which indicated that the data were normally distributed, as shown in Table 3.10.

Table 3.10 Normality Distribution of The Cognitive Load Scale Scores

Items	M	5% Trimmed Mean	SD	Skewness	Kurtosis
1	2.12	2.03	1.16	0.78	-0.26
2	2.23	2.16	1.18	0.61	-0.64
3	2.19	2.11	1.18	0.73	-0.41
4	1.92	1.81	1.09	1.19	0.57
5	1.84	1.72	1.08	1.34	1.22
6	2.10	2.01	1.15	0.84	-0.14
7	3.95	4.04	1.18	-0.90	-0.00
8	3.90	4.00	1.17	-0.92	-0.02
9	3.72	3.80	1.22	-0.73	-0.38
10	3.82	3.91	1.21	-0.83	-0.20

Then, the histograms for these items were checked, and it was found that all items 1, 2, 3, 4, 5, and 6 appeared especially positively skewed and items 7, 8, 9, and 10 were negatively skewed in their unreversed form. The reversed version of these items was found to be positively skewed. This was an expected result of the analyses considering the underlying theory of the scale. Additionally, in order to get a more distinct form of the distribution, normal probability plots were examined. The findings indicated that the instruments exhibited a normal distribution, as evidenced

by relatively straight lines. Descriptive statistics were calculated for each item and each subfactor, with the results detailed in Table 3.11.

Table 3.11 Descriptive Statistics

Item	N	Mean	Std. Deviation
Q5	757	1.84	1.08
Q4	757	1.92	1.09
Q6	757	2.10	1.15
Q1	757	2.12	1.16
Q3	757	2.19	1.18
Q2	757	2.23	1.18
Q9	757	3.72	1.30
Q10	757	3.82	1.22
Q8	757	3.90	1.18
Q7	757	3.95	1.18
Valid N (listwise)	757		

Items 5 (M=1.84) and 4 (M=1.92) received the lowest mean scores. On the other hand, items 8 (M=3.90) and 7 (M=3.95) received the highest mean scores. As for the sub-factors, the germane load was found to be the most highly endorsed dimension (M= 3.85), whereas the extraneous was the least endorsed dimension (M=1.95) among the participants (see Table 3.12).

Table 3.12 Descriptive Statistics for Subfactors

	N	Mean	Std. Deviation
Extraneous load	757	1.95	.86
Germane load	757	3.85	.95
Intrinsic load	757	2.18	.97
Valid N (listwise)	757		

Exploratory Factor Analysis

In order to evaluate the construct validity of the ten items from CLS, Exploratory Factor Analysis (EFA) was carried out. Kaiser-Meyer-Olkin (KMO) and Bartlett's Test of Sphericity were used for the assessment of sampling adequacy. Test results showed that the size of the sample was adequate (KMO=.872, Bartlett's $\chi^2(45) = 2406$, $p < .001$). Based on the original three-factor structure of the scale and scree plot (Figure 3.1), three factors were rotated using Oblique (Oblimin) rotation, which allows for correlations between factors. In this way, a three-factor solution was found, which explained 64.27% of the total variance in CLS.

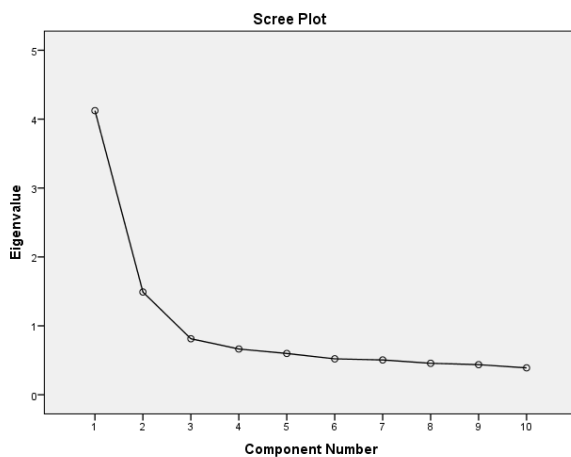


Figure 3.1. Scree Plot for CLS

Inter-factor correlation showed that the observed correlations were all less than .80, which indicated that each factor measured a unique type of cognitive load. Besides, it was observed that there was a negative correlation between IL-GL and EL-GL and a positive correlation between IL-EL (see Table 3.13). These findings aligned with the results of previous studies.

Table 3.13 Factor Correlations

	IL	EL	GL
IL	1.00	.515	-.387
EL		1.00	-.374
GL			1.00

Table 3.14 shows the factor loads of the items. All three-factor loadings of the items were greater than .60, indicating a reasonably high correlation between items and delineated factors. As seen from the table, the factor loading of items ranged between .639 and .864.

Table 3.14 Factor Loadings of the Items

Factor	Item	Factor Loading
IL	Item1	.811
	Item2	.828
	Item3	.814
EL	Item4	.864
	Item5	.789
	Item6	.639
GL	Item7	.776
	Item8	.793
	Item9	.817
	Item10	.792

Confirmatory Factor Analysis

The goodness of fit of the three-factor model obtained in EFA to the data was measured through a Confirmatory Factor analysis. Analyses were performed by using AMOS. Findings indicated that the three-factor model had acceptable goodness of fit indices: $\chi^2(32) = 68.184$, $p < 0.01$, RMSEA = 0.039, TLI = .979,

CFI= .985. As seen in Figure 3.2, the path diagram with standardized estimates of the model represents the loadings associated with each item. In addition, it can be seen that error variances did not exceed the threshold of .90.

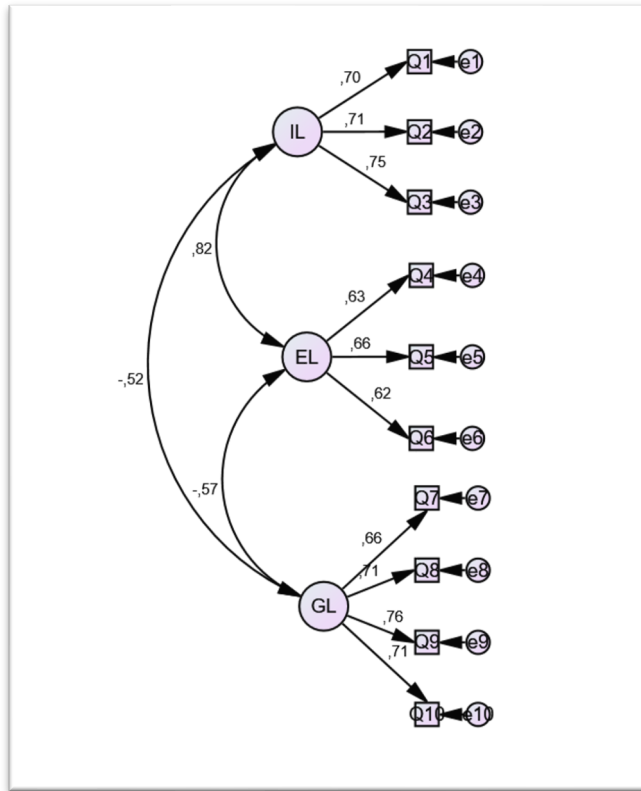


Figure 3.2. Path Analysis Diagram for CLS within CFA

3.5.2.1.3 Reliability Analysis of the Cognitive Load Scale

The instrument yielded high reliability for the overall scale ($\alpha = .84$) and two factors (intrinsic load: $\alpha = .77$; germane load: $\alpha = .81$). However, the extraneous load factor had a slightly lower α of .67 (see Table 3.15). Given the acceptable overall scale reliability, shortness of the scale and the findings from the previous studies (Hadie & Yusoff, 2016; Leppink et al., 2013; Morrison et al., 2014), it was decided to retain the extraneous load factor in the scale.

Table 3.15 Reliability Analysis Results of the CLS

	Cronbach's Alpha			
	Leppink et al., 2013	Morrison et al., 2014	Hadie & Yusoff, 2016	Current study
Intrinsic	.81	.86	.88	.77
Extraneous	.75	.85	.82	.67
Germane	.82	.93	.95	.81
Overall scale	-	.89		.84

3.5.3 Patterns of Adaptive Learning Scales (PALS)

Patterns of Adaptive Learning Scales (PALS) were developed by Midgley et al. (2000) in order to investigate the relationship between the characteristics of the learning environment and the motivation, affect, and behaviors of students. The adaptation of the instrument in Turkish language for usage in Science courses was conducted by Taş (2008) as part of a PhD thesis and the reliability of the scale was calculated as 0.81. The scale consisted of 42 items in a 5-point Likert form. This scale was adapted for the purpose of this study by replacing the term "Science" with Information Technology and Software (see Appendix D). For the new adapted version of the scale, pilot implementation was carried out with a sample of 601 sixth grade students. 82 subjects with missing values were excluded from the analysis, so reliability analysis was performed for 519 subjects. The reliability of the scale was found to be .922.

3.5.4 Attitudes Towards Coding Education Scale (ATCES)

The Attitudes Towards Coding Education scale (ATCES) was developed by (Karaman & Büyükalın Filiz, 2019) and comprises 41 items rated on a 5-point Likert scale (Appendix E). ATCES consists of two dimensions: "General Positive Attitude Towards Coding Education," which includes 28 items, and "General Negative Attitude Towards Coding Education," which includes 13 items. The response

choices on the Likert scale were established as "(1) strongly disagree, (2) disagree, (3) partly agree / partly disagree, (4) agree, (5) strongly agree". In another study using this scale, the Cronbach's alpha coefficient was reported as .956 (Özeren, 2022). The scale was administered to 420 fifth-grade students and Cronbach's alpha coefficient was found as .793, indicating a good level of reliability for the instrument. In the main study, Cronbach's alpha coefficient was found to be .941, indicating good reliability.

3.5.5 Reading Comprehension Achievement Test

The Reading Comprehension Achievement Test, developed by Kuşdemir Kayıran (2014) as part of a PhD thesis, consists of 29 items (Appendix F). In the study KR-20 reliability was determined to be .85, the average difficulty of the test was .68, and the standard deviation of the test was 5.58. Similarly, in the current study Cronbach's alpha coefficient value was found as .895.

3.5.6 5th Grade Mathematics Achievement Test

5th Grade Mathematics Achievement Test prepared by Özcan (2016) within the scope of the master's thesis (Appendix G). The four-option multiple-choice questions of the test were developed in alignment with the learning outcomes of the sub-learning areas "Natural Numbers" and "Four Operations Problems in Natural Numbers" included in the 5th-grade mathematics curriculum. The test was found to have good internal consistency with Cronbach's alpha coefficient of .88. For the current study, Cronbach's alpha coefficient value was found as .930.

3.5.7 Student Interview Protocol

The semi-structured interview protocol aimed to gather in-depth information from students regarding their perspectives on the factors that influence their coding

success. This qualitative approach provided a deeper understanding of students' experiences, beliefs, and attitudes toward coding education and its impact on their achievements. The interview protocol was subjected to a thorough review by a middle school Turkish language teacher, a university subject matter expert, and two information technologies and software teachers. The experts evaluated the protocol for any questions that were confusing, misleading or did not adequately elicit the desired information from the students. The experts provided detailed feedback on the protocol, identifying areas that needed improvement and suggesting specific revisions. According to the feedback provided by the experts, some questions were revised to enhance their clarity and grammatical structure. This ensured that the questions were easy to understand and unambiguous for students. To assess the effectiveness and understandability of the revised interview protocol, a pilot test was conducted with three fifth-grade students who were currently learning programming. Based on the feedback from the students, the researchers identified specific areas where additional questions or sub-questions could provide deeper insights into their perspectives. For instance, in response to questions about pair programming, they added sub-questions related to task switching and collaboration dynamics.

The interview protocol consisted of a total of 19 main questions, some of which have sub-questions (Appendix H). The main questions covered a broad range of topics related to coding success, including attitude, learning experiences, perceived challenges, self-efficacy beliefs, and the other factors affecting coding learning. Four questions were specifically tailored to students who have primarily engaged in paired programming throughout the semester. These questions explored the unique aspects of collaborative coding and its impact on their learning and success. Two questions were specifically designed for students who had primarily engaged in individual programming throughout the semester. These questions addressed the challenges and benefits of independent coding and its influence on their learning. Students who engaged in both paired and individual programming throughout the semester would be asked questions from both sets of questions. This allowed for a comprehensive assessment of their experiences and perspectives across different coding modalities.

3.6 Pilot Study

The pilot study was conducted in the fall semester of the 2021-2022 academic year at a public school in Rize. Due to the ongoing COVID-19 pandemic and schools transitioning to remote learning, the pilot study was also implemented remotely. The participating school was selected using purposive sampling. This method allows researchers to select participants based on their judgments to ensure the data collected is most relevant to the study's goals (Fraenkel et al., 2012). The IT teacher at the selected pilot school participated in studies for the adaptation of lesson plans and a workshop related to the selection of lesson plans to be implemented, which were part of the preparation process for this study. Consequently, he was familiar with the content of the study. The teacher's familiarity with the lesson plans became the main criterion for selecting this school. Due to the pandemic's restrictions on face-to-face meetings, the teacher's existing knowledge of the lesson plans was essential in ensuring successful implementation during the remote pilot study. There were 167 fifth-grade students enrolled in the participating school. From these, 43 students who regularly attended ITS classes and completed the required scales throughout the seven weeks were selected to participate in the pilot study. Owing to the inherent difficulties associated with facilitating unplugged activities in a remote learning environment, the originally planned ten-week pilot study was abridged to a seven-week implementation period. This modification necessitated the removal of some unplugged activities from the curriculum. Additionally, since a class period was rescheduled to 30 minutes during the remote education process, some of the existing plugged activities were also excluded from the pilot study. Basic concepts of computer programming were aimed to be covered as much as possible in the selected lesson plans. All these modifications were made after consultation with the course instructor and in accordance with his recommendations; however, it is important to note that these changes were implemented specifically for the pilot study and may not be reflected in the main study.

The objective of the pilot study was to conduct a small-scale evaluation of the proposed main study to identify any potential issues that might arise during actual implementation. Following the pilot study, discussions with the IT teacher highlighted the need to incorporate Code.org puzzles from different courses addressing various student levels. These puzzles were subsequently added to the lesson plans and arranged in a progression from easy to difficult. The final version of the programming course content, including learning objectives, corresponding lesson plans, and associated activities is presented in Table 3.16. Unplugged activities are marked with parentheses in the table. A sample lesson plan is provided in Appendix I (Week 3: Debugging with Scrat).

Table 3.16 Weekly Learning Objectives, Lesson Plans and Activities

Week	Learning objectives (Students will be able to...)	Lesson Plans and Activities
1	5.5.1.12. Explain the concept of algorithm. 5.5.1.13. Develops an algorithm for solving a problem. 5.5.2.1. Explain the basic concepts of programming.	Course F - Lesson 1: My Robotic Friends (Unplugged activity)
2	5.5.1.14. Explain flowchart components and functions. 5.5.1.15. Draw a flowchart for an algorithm. 5.5.1.16. Debug an algorithm by testing it.	I'm Changing the Flow (Unplugged activity) Rabit and carrot (Unplugged activity) If-then life of Tortop (Unplugged activity) Alas, Flowcharts Are Confused (Unplugged activity)
3	5.5.2.2. Recognize the interface and features of the block-based programming tool. 5.5.2.3. Create the appropriate algorithm to achieve the goals presented in the block-based programming environment. 5.5.2.4. Explain the structure of linear logic. 5.5.2.5. Develop an algorithm using linear logic structure.	Course 2 - Lesson 3: Maze: Sequence

Table 3.16 Weekly Learning Objectives, Lesson Plans and Activities (cont'd)

	5.5.1.16. Debug an algorithm by testing it.	Course F - Lesson 4: Debugging with Scrat
4	5.5.2.3. Create the appropriate algorithm to achieve the goals presented in the block-based programming environment. 5.5.2.5. Develop an algorithm using linear logic structure.	Course F (2018) - Lesson 5: Creating art with code
5	5.5.2.8. Explain the loop structure and its functions. 5.5.2.9. Create algorithms with loop structure.	Course 2 - Lesson 6: Maze loops Course F (2018) - Lesson 7: Drawing shapes with loops Course 2 - Lesson 7: Artist loops
6	5.5.2.8. Explain the loop structure and its functions. 5.5.2.9. Create algorithms with loop structure.	Course F (2018) - Lesson 8: Nested loops in maze Course F (2018) - Lesson 9: Nested loops with Frozen Coding with Anna and Elsa
7	5.5.1.7. Give examples of operators that can be used in problem solving. 5.5.1.10. Use operators to solve a given problem. 5.5.2.6. Explain the decision structure and its functions. 5.5.2.7. Develop algorithms with decision structures.	Wheel of conditional statements (Unplugged activity)
8	5.5.2.6. Explain the decision structure and its functions. 5.5.2.7. Develop algorithms with decision structures.	Course 3 - Lesson 7: Bee conditionals Course D (2017) - Lesson 11: Conditionals in bee Course 2 - Lesson 13: Bee conditionals
9	5.5.1.6. Explain the variables, constants and operations required to solve the problem.	Data, put it there (Unplugged activity) Course F (2018) - Lesson 14: Envelope variables (Unplugged activity)
10	5.5.1.6. Explain the variables, constants and operations required to solve the problem. 5.5.2.10. Debug the algorithms created for different structures by predicting the results of it.	Course 4 - Lesson 6: Artist variables Course F (2018) - Lesson 15: Variables with artist

3.7 Implementation of the Study

Prior to the main implementation of the study, revised lesson plans and activity sheets were shared with teachers on a website developed by the researcher (www.bikod.co). All the materials and data collection tools were provided to the teachers prior to classes, and teachers were briefly informed about the topic and the activities of the week on an instant messaging application on a weekly basis. However, the teachers were free to decide whether to teach the lessons in accordance with the provided lesson plans, provided that the topics and the learning outcomes of the weeks would not be altered. Ultimately, a 10-week implementation was conducted in the spring semester of the 2021/22 academic year in three public schools in Rize, Turkey.

At the beginning of the implementation, the Student Information Form, Mathematics Achievement Test, and Reading Comprehension Achievement Test were administered to all participating students (Table 3.17). During the implementation process, the Cognitive Load Scale was administered to all participating students at the end of each two-hour class. Throughout the implementation, the researcher took part in the lessons to observe the learning environment and gather details on the characteristics of the target group by taking notes regarding unfavorable and favorable aspects of the class, changes made in the daily lesson plans, student participation, pair-programming behaviors of the students and general remarks about the class. The researcher attended six classes each week for nine weeks. During these classes, the researcher also assisted IT teachers in handing out and collecting activity sheets and data collection tools. Additionally, getting to know the students through attendance enabled students to feel more comfortable during the one-on-one data collection procedures conducted at the end of the implementation. At the end of the implementation, interviews were held with students through semi-structured interview forms. Three students from each class were selected in line with the teachers' suggestions based on their academic achievement in the ITS course (one with low, one with moderate and one with high academic achievement). A total of

30 students were individually interviewed, and the interviews varied in length, ranging from 10 to 25 minutes.

Table 3.17 Data Collection Procedures

Duration	Data Collection Instrument
Prior to the implementation	Student Information Form
	Mathematics Achievement Test
	Reading Comprehension Achievement Test
At the end of each two-hour ITS class	Cognitive Load Scale
At the end of the implementation	Coding Achievement test
	Attitudes Towards Coding Education Scale
	Patterns of Adaptive Learning scales
	Student interview protocol

3.8 Data Analysis

3.8.1 Quantitative Data Analysis

Following the completion of the data collection phase, the data was imported into IBM's Statistical Package for Social Sciences (SPSS) software for analysis. Prior to initiating the analysis, a data screening process was undertaken to identify and address any discrepancies or errors within the dataset. To ensure data validity, the maximum and minimum values for each variable were analyzed to confirm that no values exceeded the permissible range. Descriptive statistics and inferential statistics were then used to analyze the data. For the analysis of the quantitative data, SPSS software version 26 and R Statistical Package version 4.4.1 were utilized.

To answer the first research question of the study, which explores differences in cognitive load experienced by students across different fundamental programming topics, repeated measures ANOVA was employed. To address the second and third

research questions, independent samples t-tests were employed to investigate gender and geographical school location (urban vs. suburban) differences in attitudes towards coding education, mathematics achievement, reading comprehension achievement, and coding achievement scores. Additionally, the mulrank function was used to analyze gender and geographical school location-based variations in the subscales of PALS (personal achievement goal orientations, perception of classroom goal structures, academic-related perceptions, beliefs, strategies) scores. Furthermore, to explore if there were gender and geographical school location-based differences in students' cognitive load scores across different fundamental programming topics, a doubly repeated MANOVA test was utilized. Lastly, hierarchical regression was employed to analyze how the variables used in the study predict students' achievement scores in programming.

3.8.2 Qualitative Data Analysis

The recorded interview data was transcribed using the verbatim transcription process. Subsequently, each transcript was reviewed while simultaneously listening to the corresponding audio record to ensure completeness and accuracy. The length of the interview transcripts ranged from 4 to 8 one-and-half-spaced pages. Pseudonyms were assigned to each participant, designated by the letter “S” followed by a number in the sequence of the interviews (e.g., S1, S2, etc.), and transcripts were titled with these pseudonyms. Qualitative data analysis software NVivo was employed to analyze the qualitative data.

Initially, each transcript was read several times and analytic memos were written. The process of writing analytic memos continued through the coding process to facilitate the researcher's critical reflection on the process and to reflect on “emergent patterns, categories and subcategories, themes, and concepts in the data” (Saldaña, 2009, p. 42). Coding occurred in a cyclical manner, where the researcher coded and recoded the data. According to Saldaña (2009), coding is a linking process rather than a labeling process. Data analysis was conducted using Saldaña's (2009) first-

cycle and second-cycle data analysis approaches. The coding cycles selected for the coding process, based on Saldana's Generic List are illustrated in Figure 3.3 below.

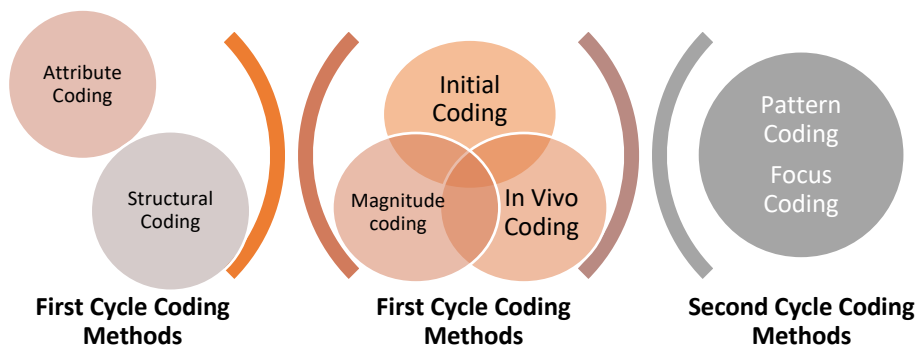


Figure 3.3. First Cycle and Second Cycle Coding Methods

In the first cycle of the coding process, a multifaceted approach was employed to extract meaning from the interview data. This approach encompassed grammatical coding methods, including attribute coding and magnitude coding. Additionally, elemental methods were utilized, including structural coding, in vivo coding and initial coding. The *attributed coding* method was utilized to categorize basic descriptive information about the participants. This method is particularly advantageous for qualitative studies involving multiple participants, as it facilitates the organization of participant data based on predefined attributes. At the outset of the study, gender, school name, and overall academic achievement status were coded for each student to establish a baseline understanding of the participants' backgrounds and academic characteristics. The *structural coding method* generally provides a basis for further detailed analysis. Interview data was segmented according to their relationship with specific research questions for further and detailed analysis. The *initial coding*, also known as open coding, method was used to break data into smaller and discrete sections. The *in vivo coding* method was employed to preserve participants' voices by utilizing their everyday language, specifically their words, terms, or phrases as codes. The *magnitude coding* was used

to denote the intensity of specific code. These coding methods were used concurrently and repeated multiple times. At the end of the first cycle coding process, a codebook was developed. In the second cycle coding, *focused coding* was employed to categorize and explain in detail the findings from the first cycle coding. Additionally, *pattern coding* was used to develop categories major and themes (Saldaña, 2009).

3.9 Trustworthiness for Qualitative Part of the Study

The credibility and transferability of findings are fundamental concerns in qualitative research, similar to reliability and validity in quantitative studies. However, due to the interpretive nature of qualitative inquiry, alternative approaches are necessary to establish these qualities effectively. To enhance the trustworthiness of this study, several strategies were employed.

3.9.1 Internal Validity (Credibility)

Prolonged Engagement and Persistent Observation in the Field

This strategy involves spending adequate time in the field to develop relationships with participants and understand the cultural context of the phenomenon, thereby facilitating the avoidance of inaccuracies caused by the researcher or informants (Creswell, 2007). In the current study, throughout the research, the researcher actively participated in all classes of six out of ten classrooms for nine weeks where the study was conducted. This involved attending two-hour classes each week. During this time, the researcher not only took field notes but also assisted the teacher in data collection activities.

Inter coder Reliability

Inter coder reliability is a statistical metric used to assess the level of agreement between multiple coders when applying a coding scheme to the same data set

(O'Connor & Joffe, 2020). Although this strategy is criticized by some researchers for relying solely on coder agreement, which may not guarantee reliable results, it is perceived as an effective method for enhancing the consistency of qualitative research findings (Merriam, 2009). Reporting intercoder reliability can help assure readers that the analysis was performed conscientiously and consistently (Kurasaki, 2000). This study employed an experienced colleague in qualitative data analysis as the second researcher. Before the coding process, the second coder was thoroughly informed about the research purpose, research questions, sample, and research process. Subsequently, all transcripts were independently coded by the researcher and the second coder. Upon completion of the coding process, the researcher and second coder met for the consistency check. Intercoder reliability was evaluated using Miles & Huberman's (1994) method $[(\text{consensus} / (\text{consensus} + \text{disagreement})) \times 100]$. For the first phase of the process, the calculation of the intercoder reliability resulted in an 85.02% agreement, indicating a high level of consistency among the coders. Subsequently, the two coders engaged in meetings to discuss the similarities and differences in their coding. These discussions led to the identification of discrepancies and the refinement of the coding scheme. For codes that were similar and expressed the same concept, a common code was determined. For codes that were different and expressed different concepts, discussions were held to decide which code would be retained. After reaching a consensus on the codes and recoding the transcripts, the result reached %93.1. This percentage surpasses the commonly accepted threshold of 90% for reliable coding (Miles & Huberman, 1994), suggesting that the coding process was dependable. For the final round, the codes were revised and organized under emergent themes.

Peer review or debriefing

Peer review or debriefing serves as another procedure for establishing credibility, involving an evaluation of the research data and methodology by someone knowledgeable about the research or the phenomenon being studied. This strategy establishes the research's credibility by offering assistance, critically examining the researchers' assumptions, encouraging them to advance methodologically, and

asking challenging questions about their methods and interpretations (Creswell & Miller, 2000). In this study, the thesis advisor closely monitored the thesis process. Additionally, the dissertation committee members were regularly informed about the progress of the study and provided suggestions and critiques related to the process.

Triangulation

Triangulation is collecting data from various sources and looking for patterns or themes that appear consistently across those sources (Creswell & Miller, 2000). Combining various methods, data sources, viewpoints, and researchers in a single study strengthens the investigation by adding depth, richness, and a broader perspective. Data source triangulation strengthens research by looking for consistency. This can involve using the same method with different data sources (e.g., interviews at different times) or comparing data from people with contrasting perspectives (Denzin, 2012). This study employed triangulation by gathering data from multiple locations and utilizing various data collection techniques. It involved the participation of fifth-grade students from three different schools. To explore the factors influencing student programming learning, the researcher implemented three distinct surveys and an interview protocol.

3.9.2 External Validity (Transferability)

Thick, rich description

External validity pertains to the extent to which research findings can be applied or generalized to a broader population, settings, or range of situations. This comprehensive description empowers readers to assess the generalizability of the findings to other situations by facilitating a critical evaluation of the extent to which the research context and participant characteristics are comparable (Cohen et al., 2017). Given the purposive sampling method employed in this study, the generalizability of the results is constrained. Nonetheless, to ensure the

transferability of the research outcomes to comparable contexts, this chapter offers an in-depth description of the sample, context, and role of the researcher.

3.9.3 Researcher Role and Bias

Researchers bring their unique experiences, beliefs, and characteristics to the study, which can be valuable for choosing the research problem, research questions, and target audience. However, these same experiences can introduce bias during analysis, affecting how data is interpreted and presented (Corbin & Strauss, 2012).

In this study, the researcher's prior experience as a middle school IT teacher and subsequent role teaching computer programming in a vocational school, coupled with beliefs about the importance and challenges of programming education, could introduce potential biases. To mitigate these potential biases and ensure the research's objectivity, the researcher implemented a series of strategies. Firstly, the researcher abstained from teaching the participating students, minimizing the impact of personal beliefs. Secondly, the researcher maintained a neutral stance during interviews to avoid influencing participant responses. Thirdly, the researcher acted as a moderator, facilitating the natural teaching process and avoiding interference with lesson plans and assessments. Finally, the interview protocol and coding achievement test underwent a rigorous process of expert review, teacher review, and pilot testing to ensure their validity. Furthermore, the researcher employed additional strategies to enhance the objectivity of the findings. A second coder was involved in the qualitative data analysis process, providing an external perspective and assisting in identifying and addressing potential biases. The second coder also revisited the coded data and the results of the analysis to refine interpretations. By implementing these proactive measures, the researcher demonstrates a commitment to minimizing bias and upholding the trustworthiness of the study. These efforts contribute significantly to the overall credibility of the research and strengthen the confidence in the findings.

3.10 Ethical Issues

Professional associations, such as the American Psychological Association's Committee on Scientific and Professional Ethics, outline crucial ethical considerations for researchers conducting human subject studies. Fraenkel et al. (2012) categorized these considerations into three key principles: “protecting participants from harm, ensuring the confidentiality of research data, and addressing potential deception of subjects” (p. 63). The researcher ensured that these ethical principles were followed throughout the research process. Prior to commencing the study, the researcher obtained the necessary approvals by submitting an application to the Middle East Technical University's Committee on Human Ethics (Appendix J and K). This application outlined the study's objectives, data collection procedures, data collection instruments, and informed consent forms. Subsequently, to collect data from middle school students in Rize, an application was submitted to the Rize Provincial Directorate of National Education, outlining the study's objectives, research process, and data collection instruments to obtain the necessary official permission (Appendix L). Additionally, approval to carry out the research was granted by the school administrations and IT teachers of the three middle schools participating in the study. Informed consent forms were used to inform the parents of the students participating in the study about the study's objectives, the confidentiality of the participants, and the potential benefits of the study. Written informed consent was obtained from the parents prior to their children's participation in the study. At the outset of the study, the purpose, aims, and all aspects of the research process, including how the results would be used, were explained to the students, and their verbal consent to participate voluntarily in the study was obtained. The students were notified that their involvement in the study was voluntary and that they had the freedom to discontinue their participation at any point. They were also informed that even if they chose to participate, they could request that their collected data not be used and be deleted at any time. Great care was taken to ensure the anonymity of the participants. During the qualitative data analysis process,

identifying information such as school, class, and student names were removed from the transcripts before sending them to the second coder. When presenting findings related to qualitative data, student codes were used instead of student names for citations. Collected data was stored in private application accounts and password-locked private devices. The data will be retained in the same format for a period of five years following the conclusion of the research, after which it will be removed.

3.11 Limitations of the Study

This study has certain limitations that should be considered when interpreting the result. Firstly, the participant schools were selected using convenience sampling, which means that they were not randomly selected from a representative population. This method could introduce selection bias, potentially restricting the generalizability of the findings to a wider population. While the study included schools from both urban and suburban areas, the data was collected from schools within the same city. Secondly, the study was limited by the scope of one online block-based programming environment (code.org) and the selected tasks in that environment, which may not cover all relevant programming skills comprehensively. Using the same learning tool throughout the semester may have been restrictive for students, especially for students with advanced programming abilities. Thirdly, the limited number of computers available forced some students to work in pairs. While paired programming has its benefits, it can also introduce challenges and disadvantages. This could have hindered some students' learning experiences and outcomes. Lastly, only written tests were used to measure student success in programming. The limited number of computers prevented the use of process-oriented or project-based assessments, especially in suburban schools. Additionally, in the implemented Code.org examples, students worked on solving problems based on the provided examples rather than creating projects from scratch. Teachers were not allowed to assign tasks that required students to develop original projects.

Written tests may not fully capture students' practical programming skills and problem-solving abilities.





CHAPTER 4

RESULTS

This chapter presents the research findings derived from both quantitative and qualitative data analyses. The findings are presented in parallel with the corresponding research questions and sub-questions.

4.1 Results of the Quantitative Data Analysis

This section presents the findings derived from the quantitative data analysis to address the corresponding research questions. First, the correlation analysis results are provided to explore the relationships between variables, followed by the presentation of statistical test results. These include descriptive statistics to present a summary of the data's characteristics, assumption tests to ensure that the chosen statistical tests met the necessary underlying assumptions, and the results and findings of the relevant statistical analysis approach.

4.1.1 Correlation Between Variables of the Study

The Pearson product-moment correlation findings for the relationship between study variables revealed that intrinsic load and extraneous load were very strongly correlated ($r = .843, p < .01$). Both intrinsic ($r = -.623, p < .01$) and extraneous load ($r = -.694, p < .01$) were negatively correlated with germane load. The relationship between intrinsic load and extraneous load with attitudes toward coding education (IL: $r = -.456, p < .01$; EL: $r = -.468, p < .01$), coding achievement (IL: $r = -.434, p < .01$; EL: $r = -.451, p < .01$), mathematics achievement (IL: $r = -.368, p < .01$; EL: $r = -.356, p < .01$) and reading comprehension achievement (IL: $r = -.253, p < .01$; EL: $r = -.297, p < .01$) were negative. While germane load positively and moderately

correlated with attitudes toward coding education ($r = .522, p < .01$), coding achievement ($r = .456, p < .01$) and mathematics achievement ($r = .434, p < .01$), the relationship between reading comprehension achievement was weak ($r = .363, p < .01$).

Upon examining the interrelationships among the subscales of the PALS and their relationships with other variables, it has been observed that MGO was positively and very strongly correlated with AE ($r = .881, p < .01$), strongly correlated with CMGS ($r = .736, p < .01$), moderately correlated with PAvGO ($r = .577, p < .01$), CPApGS ($r = .486, p < .01$) and CPAvGS. Besides that, there was a positive weak correlation between MGO and PApGO ($r = .369, p < .01$), coding achievement ($r = .291, p < .01$) and mathematics achievement ($r = .211, p < .01$). On the other hand, test results showed that MGO was negatively and strongly correlated with cheating behavior ($r = -.742, p < .01$), indicating that higher MGO is associated with lower cheating behavior. Similarly, the correlation between MGO and ASHS was negative but weak, $r = -.264, p < .01$. Additionally, test results exhibited positive and strong relationship between CMGS and CPApGS ($r = .701, p < .01$), PApGO and PAvGO ($r = .607, p < .01$), AE and CMGS ($r = .696, p < .01$), CMGS and CPAvGS ($r = .640, p < .01$), CPApGS and CPAvGS ($r = .630, p < .01$), and MA and CA ($r = .638, p < .01$). It was observed that, CPAvGS was positively and moderately correlated with PApGO ($r = .521, p < .01$), PApGO ($r = .515, p < .01$) and AE ($r = .487, p < .01$). Besides that the correlation between PApGO and CMGS ($r = .412, p < .01$), PApGO and CPApGS ($r = .400, p < .01$), PAvGO and AE ($r = .552, p < .01$), PAvGO and CMGS ($r = .497, p < .01$), AE and CPApGS ($r = .526, p < .01$), ASHS and CB ($r = .414, p < .01$), ATCE and CA ($r = .465, p < .01$), MA and RCA ($r = .586, p < .01$), RCA and CA ($r = .518, p < .01$) were positive and moderate. Test results also indicated significantly negative correlations between variables such as AE and ASHS ($r = -.249, p < .01$), AE and CB ($r = -.682, p < .01$), CB and CMGS ($r = -.560, p < .01$) and CB and CPApGS ($r = -.462, p < .01$).

Table 4.1 Correlation Coefficients Between the Variables

Variables	IL_av	EL_av	GL_av	MGO	PAPGO	PAVGO	AE	ASHS	CB	CMGS	CPAPGS	CPAVGS	ATCE	MA	RCA
EL_av	.843**														
GL_av	-.623**	-.694**													
MGO	-.149*	-.131	.047												
PAPGO	-.104	-.116	.057	.369**											
PAVGO	-.167*	-.084	.066	.577**	.607**										
AE	-.200**	-.165*	.093	.881**	.361**	.552**									
ASHS	.104	.143*	-.120	-.264**	.092	-.039	-.249**								
CB	.124	.119	-.056	-.742**	-.121	-.335**	-.682**	.414**							
CMGS	-.070	-.053	.025	.736**	.412**	.497**	.696**	-.079	-.560**						
CPAPGS	-.028	-.036	.001	.586**	.400**	.374**	.526**	-.064	-.462**	.701**					
CPAVGS	.025	-.018	.006	.486**	.521**	.515**	.487**	.031	-.312**	.640**	.630**				
ATCE	-.456**	-.468**	.522**	.122	-.012	.072	.123	-.212**	-.190**	.120	.016	.025			
MA	-.368**	-.356**	.434**	.211**	.102	.158*	.201**	-.098	-.159*	.133	.191**	.161*	.341**		
RCA	-.253**	-.297**	.363**	.030	-.016	.038	.029	-.028	-.090	.012	-.001	-.034	.356**	.586**	
CA	-.434**	-.451**	.456**	.291**	.106	.183**	.256**	-.239**	-.259**	.231**	.212**	.171*	.465**	.638**	.518**

** . Correlation is significant at the 0.01 level (2-tailed).

* . Correlation is significant at the 0.05 level (2-tailed).

(**IL_av** = Intrinsic Cognitive Load Average, **EL_av** = Extraneous Cognitive Load Average, **GL_av** = Germane Load Average, **MGO** = Mastery Goal Orientation, **PAPGO** = Performance-Approach Goal Orientation, **PAVGO** = Performance-Avoid Goal Orientation, **AE** = Academic Efficacy, **ASHS** = Academic Self Handicapping Strategies, **CB** = Cheating Behavior, **CMGS** = Classroom Mastery Goal Structure, **CPAPGS** = Classroom Performance-Approach Goal Structure, **CPAVGS** = Classroom Performance-Avoid Goal Structure, **ATCE** = Attitude Towards Coding Education, **MA** = Mathematics Achievement, **RCA** = Reading Comprehension Achievement, **CA** = Coding Achievement)

4.1.2 Results of the Research Question 1

The first research question of this study aimed to examine the differential cognitive load experienced by students across various fundamental programming concepts. A within-subjects analysis of variance (ANOVA) was conducted to examine the effect of programming concepts (basic sequences, flowcharts, testing and debugging, loops, nested loops, conditionals, and variables) on cognitive load as measured by the Cognitive Load Scale. This analysis employed a repeated-measures design with seven measurement points representing the seven topics of the programming instruction. The means, standard deviations, skewness, and kurtosis values for the observed variables are presented in Table 4.2. The students exhibited the highest mean scores for intrinsic load on the concepts of nested loops ($M = 2.131$, $SD = 1.196$), basic sequences ($M = 2.114$, $SD = 0.772$), and loops ($M = 2.003$, $SD = 1.083$), respectively. The highest extraneous load was observed for the concepts of nested loops ($M = 2.029$, $SD = 1.035$), basic sequences ($M = 1.964$, $SD = 0.687$), and flowcharts ($M = 1.945$, $SD = 0.849$). On the other hand, the mean of the students' germane load scores was lowest for basic sequences ($M = 3.706$, $SD = 0.810$), flowcharts ($M = 3.756$, $SD = 1.073$), and debugging ($M = 3.808$, $SD = 1.095$). Notably, germane load exhibited an increase across weeks for all concepts except for nested loops, where a decrease occurred when transitioning from loops to nested loops concept. Descriptive statistics revealed that skewness and kurtosis values for intrinsic, extraneous, and germane load scores across the seven programming concepts fell within the acceptable range of ± 2 , suggesting the normality of the data (Joseph F. Hair, 2021).

Table 4.2 Descriptive Statistics for Repeated Cognitive Load Measures Across Seven Programming Concepts

Dependent Variable	<i>M</i>	<i>SD</i>	Variance	Skewness	Kurtosis
IL sequences	2.114	0.772	0.595	0.399	-0.492
IL flowcharts	1.930	0.888	0.789	0.870	0.357
IL debugging	1.739	0.947	0.897	1.205	0.386
IL loops	2.003	1.083	1.173	0.936	0.156
IL nested loops	2.131	1.196	1.431	0.814	-0.364
IL conditionals	1.650	0.765	0.584	1.129	0.420
IL variables	1.658	0.814	0.662	1.185	0.588
EL sequences	1.964	0.687	0.472	0.537	-0.283
EL flowcharts	1.945	0.849	0.720	0.818	0.467
EL debugging	1.776	0.862	0.742	1.150	0.868
EL loops	1.846	0.974	0.948	1.029	0.239
EL nested loops	2.029	1.035	1.070	0.825	-0.096
EL conditionals	1.826	0.923	0.852	1.119	0.486
EL variables	1.787	0.931	0.867	1.078	0.325
GL sequences	3.706	0.810	0.656	-0.396	-0.322
GL flowcharts	3.756	1.073	1.151	-0.690	-0.256
GL debugging	3.808	1.095	1.199	-0.675	-0.446
GL loops	3.888	1.099	1.209	-0.771	-0.205
GL nested loops	3.864	1.104	1.220	-0.713	-0.351
GL conditionals	3.988	0.977	0.954	-0.854	0.115
GL variables	4.046	1.045	1.093	-1.009	0.175

To test the assumption of sphericity, the differences between each pair of measures were calculated, and their variances were compared separately for IL, EL and GL. It was observed that there was a big difference between variations of some differences. For instance, the variance of the difference between IL for conditional statements and variables was .643 while the variance of the difference between IL for flowcharts and loops were 1.385. Besides that, Mauchly's test results indicated the violation of the assumption of sphericity for IL ($\chi^2(20) = 101.24, p < .001$), EL ($\chi^2(20) = 79.29, p < .001$) and GL ($\chi^2(20) = 125.98, p < .001$). Therefore, Greenhouse-Geisser correction was used for three measures ($\epsilon = .85$ for IL, $\epsilon = .88$ for EL and $\epsilon = .86$ for GL) (Field, 2005).

Separate one-way repeated-measures ANOVAs were employed for each dependent variable (IL, EL, and GL). Test results revealed significant effects of programming concepts on intrinsic load ($F(5.12, 1013.57) = 15.06, p < .001$, multivariate $\omega^2 = .07$), extraneous load ($F(5.30, 1049.56) = 4.38, p = .001$, multivariate $\omega^2 = .02$), and germane load ($F(5.02, 994.69) = 5.03, p < .001$, multivariate $\omega^2 = .03$) (Table 4.3). These findings indicated significant variations in all three types of cognitive load scores across the seven basic programming concepts. Findings suggested a medium effect size for intrinsic load and small effect sizes for extraneous and germane load (Pallant, 2016).

Table 4.3 Results of One-way Repeated Measures ANOVA Comparing Cognitive Load Scores Across Seven Different Programming Concepts

Variable	<i>df</i>	<i>F</i>	<i>p</i>	η^2
Intrinsic Load	5.12	15.06	.000	.07
Extraneous Load	5.30	4.38	.000	.02
Germane Load	5.02	5.03	.000	.03

The results of follow-up pairwise comparison using Holm's sequential Bonferroni procedure to control for type 1 error revealed that students exhibited higher intrinsic load when learning basic sequences ($p < .001$), flowcharts ($p < .005$), loops ($p = .001$), and nested loops ($p < .001$) compared to conditionals and variables. Similarly, the intrinsic load was higher for basic sequences ($p < .001$), loops ($p < .05$), and nested loops ($p < .001$) compared to testing and debugging. Examination of the extraneous load showed that students experienced significantly higher levels of extraneous load during the learning process of basic sequences compared to testing and debugging ($p < .05$). Additionally, students exhibited significantly higher extraneous load when learning nested loops compared to learning concepts of testing and debugging ($p < .05$), conditionals ($p < .005$), and variables ($p < .05$). However, regarding germane load, it was observed that during lessons focused on variables, students demonstrated significantly higher germane load compared to learning basic sequences ($p < .001$) and flowcharts ($p < .05$). Similarly, when conditionals were taught, students exhibited significantly higher germane load compared to learning

basic sequences ($p < .001$). These findings suggest that certain programming concepts, such as basic sequences, loops, and nested loops, impose a higher cognitive load on students regarding both intrinsic and extraneous load. Moreover, the emphasis on certain concepts, such as variables and conditionals, seems to facilitate higher levels of germane cognitive processing among students.

4.1.3 Results of the Research Question 2

The second research question of this study aimed to explore whether students' adaptive learning patterns, attitudes toward coding education, cognitive load, and achievement in programming vary according to gender. To further investigate RQ1, three sub-questions were examined

4.1.3.1 Results of the Sub-Research Question 2a

Analysis of variance (ANOVA) is a statistical method used to compare groups based on a single dependent variable. Multivariate analysis of variance (MANOVA) is an extension of the ANOVA. MANOVA is used when there are multiple dependent variables to evaluate the statistical differences between dependent variables based on the independent grouping variable. The dependent variables should exhibit a conceptual association or possess a rationale justifying for being considered together (Pallant, 2016). To conduct the MANOVA test, there are some assumptions to be met. In the current study to test the multivariate normality, Mardia's measure of multivariate kurtosis was implemented using AMOS software. Mardia's multivariate kurtosis value was found to be 8.03. The critical ratio (c.r.) for kurtosis was 4.48. The significant result (c.r. = 4.48, $p < 0.05$) suggested that the data did not follow a multivariate normal distribution. Therefore, a robust MANOVA test was carried out on the ranked data using Munzel & Brunner's (2000) method to examine the gender differences in PALS (MGO, PApGO, PAvGO, CMGS, CPApGS, CPAvGS, AE, CB, and ASHS) scores of the students. The analysis was conducted in the R

Statistical Package using the ‘mulrank()’ function from the WRS package. Since the newest version of the WRS2 package does not contain this function, the analysis was conducted utilizing the original WRS package (Field, et al., 2016). Test results indicated that differences between female and male students on the dependent measures were statistically nonsignificant, $F = 1.28, p = .28$.

4.1.3.2 Results of the Sub-Research Questions 2b, 2c, 2d, and 2e

Separate independent-samples t -tests were conducted to evaluate whether there was a significant difference in attitude toward coding education, mathematics achievement, reading comprehension achievement, and coding achievement scores between males and females. The results indicated no significant differences in attitudes towards coding education ($t(197) = 0.57, p > .05$), mathematics achievement ($t(184.08) = -0.02, p > .05$), reading comprehension achievement ($t(197) = -0.89, p > .05$), or coding achievement ($t(197) = 0.30, p = .76$) scores between males and females (Table 4.4).

Table 4.4 Results of t-test and Descriptive Statistics for ATCE, MA, RCA and CA by Gender

	Males		Females		t	p	Cohen's d
	M	SD	M	SD			
ATCE	153.09	27.47	150.86	27.27	0.57	.57	0.080
MA	15.49	6.08	15.51	6.82	-0.22	.98	0.003
RCA	13.16	5.98	13.90	5.71	-0.89	.37	0.130
CA	50.56	10.20	50.14	9.31	0.30	.76	0.040

4.1.3.3 Results of the Sub-Research Question 2f

A doubly multivariate repeated-measures MANOVA was conducted on measures of three types of cognitive load: intrinsic load, extraneous load, and germane load, across seven topics within computer programming. This statistical technique is particularly suited for studies involving multiple dependent variables measured

repeatedly at different time points. The doubly multivariate repeated-measures MANOVA was employed in this study to investigate the multivariate main effects of programming topics and gender on cognitive load.

The descriptive statistics of the observed variables over all combinations of gender and programming topics are presented in Table 4.5. The intrinsic load mean score of the females ($M = 3.77$, $SD = 1.01$) and males ($M = 3.75$, $SD = 1.13$) for conditionals was the highest among all seven topics covered in the introductory programming fundamentals. Both females ($M = 3.97$, $SD = 1.07$) and males ($M = 3.82$, $SD = 1.13$) had the highest mean scores for extraneous load on nested loops than on any other subjects. The mean values of germane load scores revealed that female students exhibited the highest germane load when dealing with variables ($M = 4.07$, $SD = 0.97$), while male students showed the highest germane load when handling loops ($M = 4.06$, $SD = 0.96$).

Despite unequal female and male group sizes, the analysis proceeded because both groups were large, had more cases than dependent variables, and exhibited no significant size discrepancy (Tabachnick & Fidell, 2012). Skewness and kurtosis values for the three cognitive load types across seven programming concepts in both female and male participants supported the assumption of normality. All variables exhibited skewness and kurtosis values within the range of ± 2 . Furthermore, the highest Cook's distance value for each measurement was below the commonly accepted threshold of 1.0, suggesting the absence of significant outliers (Field, 2005). Correlation analysis revealed that Pearson's correlation coefficients between all pairs of Cognitive Load measures across the seven topics did not exceed 0.80. Therefore, the assumption of multicollinearity was established.

Box's M Test of Equality was significant ($F(231,107831) = 1.53$, $p < .001$), suggesting a departure from the assumption of homogeneity of variance-covariance matrices. This finding aligns with the increased risk of alpha-level distortion associated with a larger number of dependent variables, as noted by Tabachnick & Fidell (2012). In this study, with twenty-one dependent variables, such an outcome

is not entirely unexpected. However, further examination of individual variable variances within each group demonstrated minimal variance disparity across groups. No variable exhibited a largest-to-smallest variance ratio approaching 10:1, suggesting a limited impact on the analysis. Consequently, the analysis can proceed to the next step with relative confidence. Given the potential for assumption violation, Pillai's Trace was employed instead of Wilks' Lambda to evaluate multivariate significance due to its robustness (Tabachnick & Fidell, 2012). Thus, the evaluation of assumptions for the doubly-multivariate analysis of variance (dMANOVA) yielded acceptable results.

Table 4.5 Descriptive Statistics for Cognitive Load Measures for Gender Across Seven Programming Concepts

	Females (N = 92)				Males (N = 107)			
	<i>M</i>	<i>SD</i>	Skewness	Kurtosis	<i>M</i>	<i>SD</i>	Skewness	Kurtosis
IL sequences	2.22	0.79	0.27	-0.48	2.02	0.75	0.49	-0.45
IL flowcharts	1.96	0.64	0.16	-0.68	1.97	0.73	0.75	-0.14
IL debugging	3.72	0.80	-0.39	-0.24	3.70	0.82	-0.40	-0.35
IL loops	1.88	0.82	1.00	1.16	1.97	0.95	0.77	-0.09
IL nested loops	1.89	0.77	0.56	-0.48	1.99	0.92	0.90	0.66
IL conditionals	3.77	1.01	-0.72	0.17	3.75	1.13	-0.67	-0.51
IL variables	1.86	1.01	0.95	-0.37	1.63	0.88	1.49	1.51
EL sequences	1.84	0.86	1.02	0.71	1.72	0.87	1.29	1.19
EL flowcharts	3.85	0.95	-0.55	-0.42	3.78	1.21	-0.68	-0.66
EL debugging	2.14	1.12	0.82	0.17	1.89	1.04	1.06	0.21
EL loops	1.86	0.98	1.19	1.12	1.84	0.98	0.90	-0.46
EL nested loops	3.97	1.07	-0.96	0.29	3.82	1.13	-0.64	-0.47
EL conditionals	2.10	1.17	0.89	0.03	2.16	1.23	0.76	-0.61
EL variables	1.99	0.92	0.68	-0.23	2.07	1.12	0.86	-0.22
GL sequences	3.99	1.05	-0.90	0.09	3.76	1.15	-0.57	-0.58
GL flowcharts	1.71	0.76	0.95	-0.01	1.60	0.77	1.32	0.98
GL debugging	1.83	0.88	1.26	1.32	1.82	0.96	1.04	0.01
GL loops	3.90	1.00	-0.81	0.24	4.06	0.96	-0.91	0.05
GL nested loops	1.63	0.75	1.39	1.71	1.68	0.87	1.06	0.001
GL conditionals	1.79	0.85	1.17	1.32	1.78	1.00	1.03	-0.18
GLL variables	4.07	0.97	-1.09	0.64	4.02	1.11	-0.95	-0.10

As presented in Table 4.6, the doubly repeated MANOVA revealed a non-significant multivariate main effect for the interaction between gender and programming topics, Pillai's $V = .139$, $F(18, 180) = 1.62$, $p > .05$, partial $\eta^2 = .139$. The interaction effect

indicates that different programming topics had no different effects on males and females in terms of types of cognitive load. Besides that, there was a statistically significant multivariate main effect for programming topics, Pillai's $V = .424$, $F(18, 180) = 7.35$, $p < .001$, partial $\eta^2 = .424$ with a large effect size. There was not a statistically significant multivariate main effect for gender, Pillai's $V = .021$, $F(3, 195) = 1.39$, $p = .021$, partial $\eta^2 = .021$. This finding indicates that there were no significant differences between male and female students in terms of intrinsic load, extraneous load, and germane load across the seven topics within computer programming.

Table 4.6 Results of Doubly Repeated MANOVA for Cognitive Load Types by Gender

Multivariate Effect		Pillai's V	F	df	Error df	p	Partial η^2
Between Subjects	Gender	.021	1.39	3	195	.246	.021
Within Subjects	Programming topics	.424	7.35	18	180	.000	.424
	Interaction	.139	1.62	18	180	.059	.139

Follow-up univariate ANOVAs were then examined, and it was observed that Mauchly's test revealed the violation of the assumption of sphericity for IL ($\chi^2(20) = 101.04$, $p < .001$), EL ($\chi^2(20) = 78.76$, $p < .001$) and GL ($\chi^2(20) = 131.04$, $p < .001$). Consequently, the degrees of freedom were adjusted using Greenhouse-Geisser estimates of sphericity ($\varepsilon = .85$ for IL, $\varepsilon = .88$ for EL, and $\varepsilon = .83$ for GL). Univariate test results revealed that intrinsic load differed significantly across topics, $F(5.11, 1006.80) = 15.06$, $p < .001$, partial $\eta^2 = .071$. Similarly, extraneous load and germane load also showed significant differences across topics, $F(5.30, 1044.32) = 4.16$, $p = .001$, partial $\eta^2 = .021$, and $F(5.00, 984.04) = 4.94$, $p < .001$, partial $\eta^2 = .024$, respectively. These results suggested that the levels of intrinsic, extraneous, and germane load vary depending on the topic.

4.1.4 Results of the Research Question 3

This research question of this study investigated whether there were differences in students' adaptive learning patterns, attitudes toward coding education, cognitive load, and achievement in programming based on geographical school location. To address this question, three sub questions were further investigated.

4.1.4.1 Results of the Sub-Research Question 3a

Since the multivariate normality assumption was violated for the subscales of the PALS, as previously indicated, a robust non-parametric version of MANOVA was conducted in R through mulrank function using Munzel & Brunner's (2000) method to examine the effect of gender on the scores of the students from the subscales of the PALS. The results indicated that there were statistically significant differences based on geographical school location, $F = 19.38, p < .001$. To determine which subscales showed significant differences based on geographical school location, follow-up analyses were conducted using the Mann-Whitney U test. Test results showed that there were significant differences between students from the school in the urban area and from the suburban area in all nine subscales of PALS (Table 4.7). Urban students exhibited significantly higher average ranks compared to their suburban counterparts in the following subscales: MGO ($z = -5.68, p < .001$), PApGO ($z = -2.66, p < .05$), PAvGO ($z = -3.38, p = .001$), CMGS ($z = -4.76, p < .001$), CPApGS ($z = -5.22, p < .001$), CPAvGS ($z = -3.40, p = .001$), and AE ($z = -5.82, p < .001$). On the other hand, suburban students scored significantly higher in ASHS ($z = -2.98, p < .05$) and CB ($z = -4.88, p < .001$).

Table 4.7 Mann-Whitney U Test Results for PALS by Geographical School Location

Variable	Urban ($n = 112$)	Suburban ($n = 97$)	U	z	p
	Mean Rank	Mean Rank			
MGO	120.38	73.76	2589.50	-5.68	.000
PApGO	109.54	87.71	3803.00	-2.66	.008
PAvGO	112.13	84.39	3514.00	-3.38	.001
CMGS	117.08	78.01	2958.50	-4.76	.000
CPApGS	118.72	75.90	2775.50	-5.22	.000
CPAvGS	112.53	83.87	3469.00	-3.40	.000
AE	120.91	73.09	2530.50	-5.82	.000
ASHS	89.31	113.76	3675.00	-2.98	.003
CB	82.82	122.12	2947.50	-4.88	.000

4.1.4.2 Results of the Sub-Research Questions 3b, 3c, 3d, and 3e

Separate independent-sample t -tests were conducted to compare the attitude toward the coding education scale, mathematics achievement test, reading comprehension achievement test, and coding achievement test scores of the students from urban and suburban schools. The results indicated significant differences in the MA scores ($t(197) = 3.37, p = .001$) and CA scores ($t(161.93) = 3.68, p < .001$) between students from urban and suburban schools (Table 4.8). The effect size, as measured by Cohen's d , indicated a small effect size for mathematics achievement and a medium effect size for coding achievement. These results suggest that students from urban schools had significantly higher coding scores compared to students from suburban schools. Similarly, the mathematics scores of the students from urban schools were higher than the scores of the students from suburban schools. However, the results showed that there was no significant difference in the ATCE ($t(197) = 1.04, p > .05$) and RCA ($t(197) = -2.00, p = .05$) between urban and suburban schools.

Table 4.8 Results of t-test and Descriptive Statistics for ATCE, MA, RCA and CA by Geographical School Location

	Urban (n = 112)		Suburban (n = 87)		<i>t</i>	<i>p</i>	Cohen's <i>d</i>
	<i>M</i>	<i>SD</i>	<i>M</i>	<i>SD</i>			
ATCE	153.84	26.13	149.77	28.80	1.04	.30	0.15
MA	16.81	6.38	13.80	6.01	3.37	.001	0.49
RCA	12.78	5.87	14.44	5.74	-2.00	.05	0.29
CA	52.60	8.48	47.49	10.59	3.78	.000	0.53

4.1.4.3 Results of the Sub-Research Questions 3f

The doubly multivariate repeated-measures MANOVA was employed to investigate the multivariate main effects of programming topics and geographical school location on intrinsic load, extraneous load, and germane load. The descriptive statistics of the observed variables over all combinations of school location and programming topics are presented in Table 4.9. The descriptive statistics of cognitive load measures across seven programming concepts revealed distinct patterns between urban ($N = 108$) and suburban ($N = 86$) students. In urban settings, the highest mean values for intrinsic load (IL) were observed in sequences ($M = 2.03$, $SD = 0.75$) and nested loops ($M = 1.87$, $SD = 1.05$), whereas suburban students showed higher means in nested loops ($M = 2.38$, $SD = 1.22$) and sequences ($M = 2.22$, $SD = 0.79$). For extraneous load (EL), urban students had the highest means in sequences ($M = 1.80$, $SD = 0.68$) and nested loops ($M = 1.83$, $SD = 0.99$), while suburban students exhibited higher means in nested loops ($M = 2.19$, $SD = 0.99$) and flowcharts ($M = 2.09$, $SD = 0.83$). Regarding germane load (GL), urban students reported the lowest means in sequences ($M = 3.84$, $SD = 0.77$) and debugging ($M = 3.084$, $SD = 1.15$), compared to suburban students who showed the lowest means in flowcharts ($M = 3.56$, $SD = 1.09$) and sequences ($M = 3.60$, $SD = 0.81$).

Table 4.9 Descriptive Statistics for Cognitive Load Measures for School Location Across Seven Programming Concepts

	Urban (n = 112)				Suburban (n = 87)			
	<i>M</i>	<i>SD</i>	Skewness	Kurtosis	<i>M</i>	<i>SD</i>	Skewness	Kurtosis
IL sequences	2.03	0.75	0.434	-0.486	2.22	0.79	0.353	-0.443
IL flowcharts	1.76	0.83	0.931	0.183	2.10	0.87	0.740	0.486
IL debugging	1.60	0.90	1.550	1.359	1.88	0.94	0.841	-0.359
IL loops	1.81	1.07	1.213	0.543	2.17	1.04	0.753	0.260
IL nested loops	1.87	1.05	1.036	0.134	2.38	1.22	0.507	-0.746
IL conditionals	1.59	0.73	1.518	2.075	1.69	0.77	0.828	-0.579
IL variables	1.61	0.87	1.406	1.061	1.74	0.75	0.794	-0.281
EL sequences	1.87	0.68	0.812	0.291	2.06	0.66	0.248	-0.450
EL flowcharts	1.80	0.84	1.128	1.145	2.09	0.83	0.621	0.443
EL debugging	1.71	0.87	1.240	0.742	1.79	0.73	0.640	-0.289
EL loops	1.71	1.01	1.352	0.834	1.97	0.85	0.504	-0.770
EL nested loops	1.83	0.99	1.151	0.470	2.19	0.99	0.472	-0.386
EL conditionals	1.78	0.97	1.275	0.573	1.79	0.74	0.532	-0.900
EL variables	1.79	1.06	1.156	0.170	1.80	0.78	0.675	-0.475
GL sequences	3.84	0.77	-0.320	-0.525	3.60	0.81	-0.412	-0.263
GL flowcharts	3.99	0.98	-0.882	0.287	3.56	1.09	-0.548	-0.437
GL debugging	3.84	1.15	-0.782	-0.302	3.78	1.01	-0.519	-0.579
GL loops	4.05	1.06	-1.064	0.634	3.78	1.04	-0.395	-0.956
GL nested loops	3.94	1.16	-0.953	-0.081	3.80	1.02	-0.365	-0.671
GL conditionals	4.01	0.98	-0.831	-0.171	4.05	0.88	-0.691	-0.101
GL variables	4.11	1.06	-1.117	0.326	4.05	0.94	-0.901	0.271

Although sample sizes were not equal between urban and suburban groups, there were no significant size discrepancy and both groups had more cases than dependent variables (Tabachnick & Fidell, 2012). To assess the normality assumption, skewness and kurtosis values were examined for the three cognitive load types across seven programming concepts in both urban and suburban schools. The analysis indicated that all variables exhibited skewness and kurtosis values within the acceptable range of ± 2 , supporting the assumption of normality. Additionally, the maximum Cook's distance value was below the commonly accepted threshold of 1.0 for each measurement, indicating that there were no significant outliers influencing the results (Field, 2005). Correlation analysis indicated that Pearson's correlation coefficients between all pairs of Cognitive Load measures across the seven topics were below 0.80. Consequently, the assumption of multicollinearity was not

violated. The Box's M Test of Equality ($F(28, 119003) = 1.43, p > .05$) revealed a nonsignificant result, suggesting that the assumption of homogeneity of variance-covariance matrices was met. Therefore, it can be concluded that the assumptions for the dMANOVA were satisfactorily met.

As presented in Table 4.10, the doubly repeated MANOVA revealed a nonsignificant multivariate main effect for the interaction between school location and programming topics, Pillai's $V = .126, F(18, 180) = 1.44, p > .05$, partial $\eta^2 = .126$. There was a statistically significant multivariate main effect for programming topics, Pillai's $V = .436, F(18, 180) = 7.75, p < .001$, partial $\eta^2 = .436$, indicating a large effect size. Similarly, test results revealed a significant main effect for geographical school location, Pillai's $V = .048, F(3, 195) = 3.275, p < .05$, partial $\eta^2 = .048$, with a small effect size.

Follow-up univariate ANOVAs were then examined, and it was observed that Mauchly's test revealed that the assumption of sphericity was violated for IL ($\chi^2(20) = 99.76, p < .001$), EL ($\chi^2(20) = 77.89, p < .001$) and GL ($\chi^2(20) = 124.82, p < .001$). Consequently, the degrees of freedom were adjusted using Greenhouse-Geisser estimates of sphericity ($\epsilon = .89$ for IL, $\epsilon = .92$ for EL, and $\epsilon = .87$ for GL).

Table 4.10 Results of Doubly Repeated MANOVA for Cognitive Load Types by Geographical School Location

Multivariate Effect		Pillai's V	F	df	Error df	p	Partial η^2
Between Subjects	School location	.048	3.28	3	195	.022	.048
Within Subjects	Programming topics	.436	7.75	18	180	.000	.436
	Interaction	.126	1.44	18	180	.118	.126

Univariate test results revealed that intrinsic load differed significantly across topics, $F(5.13, 1010.36) = 15.86, p < .001$, partial $\eta^2 = .075$. Similarly, extraneous load and germane load also showed significant differences across topics, $F(5.30, 1043.77) = 5.07, p = .001$, partial $\eta^2 = .025$, and $F(5.03, 991.25) = 5.58, p < .001$, partial $\eta^2 = .028$, respectively. These results suggest that the levels of intrinsic, extraneous, and

germane load vary depending on the topic, similar to the results of the doubly repeated MANOVA test conducted to examine the effects of gender and programming topics on three types of cognitive load.

When the results for geographical school location were examined, the test of between-subjects effects indicated that the only statistically significant difference, using a Bonferroni adjusted alpha level of .017, was obtained for the intrinsic load ($F(1, 197) = 7.77, p = .006$, partial $\eta^2 = .038$). On the other hand, differences between students from suburban schools and urban schools on the EL ($F(1, 197) = 2.34, p = .128$ partial $\eta^2 = .012$) and GL ($F(1, 197) = 1.28, p = .260$ partial $\eta^2 = .006$) variables were not statistically significant.

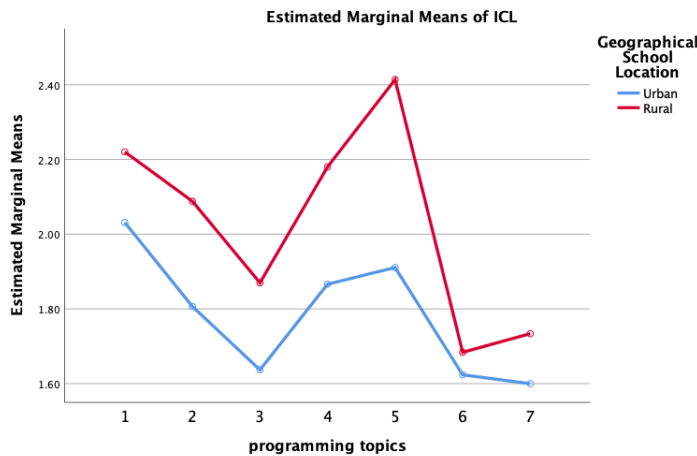


Figure 4.1 Plot of Estimated Marginal Means of Intrinsic Load by Gender

Post-hoc tests using Bonferroni correction revealed that the intrinsic load of the students from suburban schools ($M = 2.09, SD = 0.09$) was significantly higher than the students from urban schools ($M = 1.81, SD = 0.08$) for the topics of flowcharts ($p = .026$). Similarly, during the week when the topic of loops was covered, students from suburban schools reported significantly higher intrinsic cognitive load ($M = 2.18, SD = 0.12$) compared to students from urban schools ($M = 1.87, SD = 0.10, p < .04$). Another topic that suburban students experienced higher intrinsic load ($M = 2.41, SD = 0.13$) than the urban students ($M = 1.91, SD = 0.11$) was nested loops ($p = .003$), as seen in Figure 4.1.

4.1.5 Results of the Research Question 4

The fourth research question of the study aimed to explore the predictive power of research variables in relation to students' coding achievement scores. Hierarchical regression was conducted to test the fourth hypothesis to determine the relative strength of the dependent variables in predicting the coding achievement scores of the students. The regression analysis followed the 4-stage process. In the analysis, demographic variables (gender and geographic school location) were initially introduced to the model. Subsequently, mathematics achievement, reading comprehension achievement, extraneous load, germane load, attitude, and academic efficacy variables, which have been discussed and evidenced in the related literature to be associated with programming success, were included in the model. As a third step, performance-approach goal orientation and performance-avoidance goal orientation variables were introduced to the model. Finally, exploratory environmental factors (classroom mastery goal structure, classroom performance-approach goal structure, and classroom performance-avoid goal structure), cheating behavior and academic self-handicapping strategies were incorporated into the model. The order of variable entry into the regression equation was determined by considering studies in the theoretical framework and related literature that examined variables associated with the outcome variable.

Prior to conducting hierarchical regression, the underlying assumptions of the analysis were ascertained. Initially, the minimum sample size was determined by considering a statistical power of 0.80 and an alpha level of 5%. The effect size chosen for the multiple hierarchical regression design with fifteen dependent variables was set at 0.20. Employing a G*Power calculator, it was determined that the present study required 108 participants. Besides that, the minimum sample size required for hierarchical regression analysis was calculated by using the formula $[50 + 8 \times 15]$ suggested by Tabachnick & Fidell (2012) and found to be 170. Based on the aforementioned considerations, it can be concluded that the sample size for the analysis was adequate.

To evaluate the multicollinearity, the correlations between the predictor variables were examined. The correlation coefficient values between each variable were below .80 (Field, 2005), except between MGO and AE ($r = 0.881$) and between IL and EL ($r = 0.843$), as previously mentioned (p. 87). Concerns regarding multicollinearity potentially biasing the regression model led to the removal of one of the highly correlated variables. Based on the variance inflation factor (VIF) and tolerance indices, MGO was chosen for exclusion from the analysis due to its higher VIF and lower tolerance indices compared to the AE variable. Adopting a similar approach, it was decided to remove IL from the analysis as well. After the exclusion of these two variables, tolerance indices and VIF statistics were checked for multicollinearity (Table 4.11). The analysis results showed that for all predictor variables, the tolerance indices were above the threshold value of .20 (Menard, 2010, p.76), and all VIF values were below the threshold value of 5. Additionally, the average VIF was calculated as 2.103, which is not substantially greater than the suggested value of 1 (Field, 2005). Therefore, the multicollinearity assumption was met.

Table 4.11 Collinearity Statistics of the Predictor Variables

Variables	TI	VIF
Geographic school location	0.647	1.545
Gender	0.911	1.098
MAT	0.493	2.029
RCT	0.525	1.904
EL	0.480	2.081
PSL	0.436	2.295
AE	0.308	3.244
ATCE	0.625	1.601
PApGO	0.524	1.907
PAvGO	0.481	2.081
CMGS	0.308	3.243
CPApGS	0.405	2.470
CPAvGS	0.440	2.270
HANDI	0.748	1.337
CHEAT	0.409	2.444

To assess the independence of residuals, the Durbin-Watson statistic was employed. The test result of 2.255, falling within the range of 1 and 3, suggested that the assumption of independence was met. Standardized residual statistics were assessed to identify potential outliers. The analysis revealed eight cases with standardized residuals falling within the range of -2 to +2 (Table 4.12). As anticipated in a normal distribution, approximately 95% of the cases are expected to exhibit standardized residuals within this range. Consequently, the presence of eight cases (less than 10% of the sample) with standardized residual values outside these limits is not considered a significant concern. Additionally, Mahalanobis distances were checked to evaluate multivariate outliers (Pallant, 2016). Based on the critical Chi-Square values table, critical χ^2 at a significance level (α) of .001 for fifteen degrees of freedom is 37.7. The test results showed that the maximum Mahalanobis distance value was 37.186. Besides that, the maximum Cook's distance value was 0.059, which is below 1 (Field, 2005). These findings indicated the absence of multivariate outliers for all independent variables.

Table 4.12 Standardized Residual Statistics

Case Number	Std. Residual	CA	Predicted Value	Residual
1	-2.912	19.5	39.0288	-19.52876
3	-2.701	21.75	39.8652	-18.11517
5	-2.720	40.5	58.7444	-18.24440
6	-2.796	17.5	36.2546	-18.75457
12	-2.067	34	47.8630	-13.86300
15	-2.037	22.5	36.1651	-13.66513
20	-2.678	33.5	51.4590	-17.95896
21	2.200	61	46.2414	14.75857

Note. For the dependent variable coding achievement (CA)

To test the normality of the residual assumption, the probability plot (P- P) and histogram of the regression standardized residual were examined. As seen in Figure

4.2, residuals had a straight-line relationship with predicted CA scores, indicating no major deviations from normality. Similarly, the histogram of the regression standardized residual showed a roughly normal distribution for CA scores (Figure 4.3).

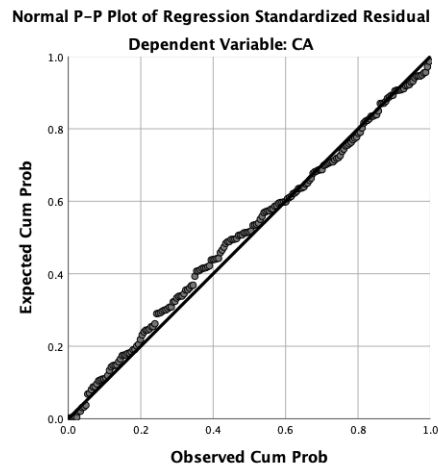


Figure 4.2 Normal Probability Plot (P- P) of the Regression Standardized Residual for CA

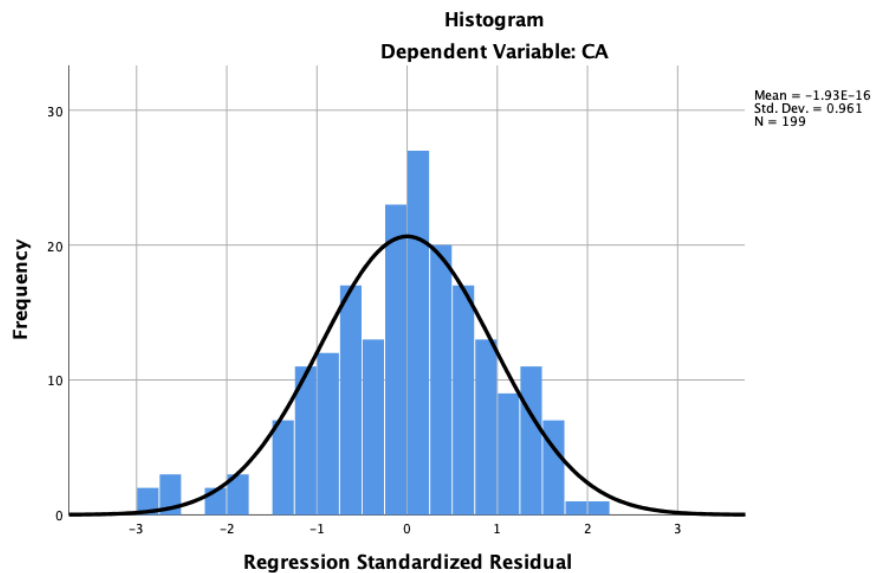


Figure 4.3 Histogram of Regression Standardized Residual for CA

The homoscedasticity assumption was assessed through a scatterplot of standardized residuals and standardized predicted values (ZRESID vs. ZPRED). Examination of Figure 4.4 revealed a random scatter of residuals, confirming the assumption of homoscedasticity.

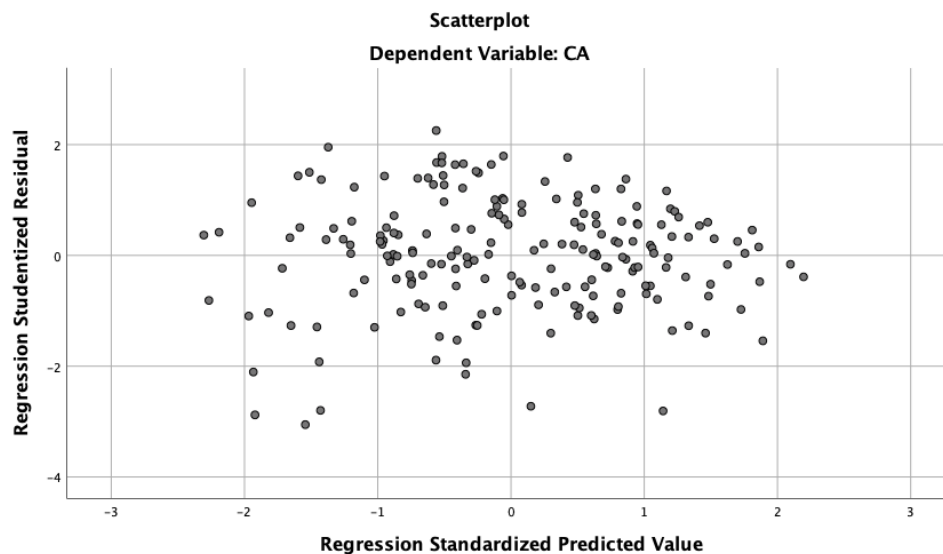


Figure 4.4 Scatterplot of Standardized Residuals and Standardized Predicted Values

To assess the linearity assumption, scatterplots were examined between each predictor variable and the dependent variable. The scatterplots demonstrated a linear relationship between the predictors and the dependent variable. Besides that, the linearity tests indicated a linear relationship between the variables. Partial plots were checked for the homoscedasticity and linearity assumptions. While a strong correlation between each predictor variable and the outcome variable was not observed, examination of the scatterplots indicated a lack of prominent outliers. Moreover, the data points were evenly dispersed around the regression line, implying homoscedasticity. Additionally, the graphs did not show any pattern indicating there is no violation of both the assumption of linearity and homoscedasticity (Field, 2005).

Hierarchical regression was conducted to predict the overall coding achievement score from fifteen predictor variables. Demographic variables (age and geographic school location) were introduced to the model in Step 1, explaining 7% of the variance in coding achievement scores, $R^2 = .07$, $F(2, 196) = 7.20$, $p = .001$. Results indicate that geographic school location was a significant predictor of coding achievement, $B = -5.124$, $\beta = -.261$, $t = -3.781$, $p < .001$. This means that academic achievement scores decreased significantly more in suburban schools compared to urban schools. Conversely, gender was not a significant predictor of coding achievement ($R^2 = .07$, $F(2, 196) = 7.20$, $p = .001$), which means that there was no statistically significant difference in predicting the coding achievement scores of participating students who were girl compared to those students who were boy.

Six variables (mathematics achievement, reading comprehension achievement, extraneous load, germane load, academic efficacy, and attitude towards coding achievement) were added to the model in the second step and test results showed that these variables account for an additional 48% variance in coding achievement controlling for gender and geographic school location, $R^2 \text{ change} = .48$, $F \text{ change}(2, 190) = 33.19$, $p < .001$. This means that nearly half of the variance in coding achievement was accounted for by the added variables. Analysis results for Model 2 indicated that mathematics achievement ($B = 0.50$, $\beta = .33$, $t = 4.84$, $p < .001$), reading comprehension achievement ($B = 0.39$, $\beta = .24$, $t = 3.53$, $p = .001$), extraneous load ($B = -0.70$, $\beta = -.14$, $t = -2.06$, $p < .05$), and attitude toward coding education ($B = 0.06$, $\beta = .17$, $t = 2.81$, $p < .01$) were statistically significant predictors of coding achievement. Among these variables, the majority of the variance in the coding scores was uniquely explained by mathematics achievement scores ($sr = .24$) and followed by reading comprehension achievement ($sr = .17$), attitude toward coding education ($sr = .14$), extraneous load ($sr = .10$), respectively. These results showed that the coding achievement scores of the students were higher when their performance in mathematics and reading comprehension, as well as their attitude towards coding education and extraneous load were higher. On the other hand, analysis results showed that germane load ($B = 0.08$, $\beta = .02$, $t = 0.33$, $p > .05$) and

academic efficacy ($B = 0.10, \beta = .06, t = 1.07, p > .05$) were not significant predictors of the output variable.

In the third step, two variables (performance-approach goal orientation and performance-avoidance goal orientation) were added to the model. The test results showed that including performance-approach goal orientation ($B = -0.02, \beta = -.01, t = -0.18, p > .05$) and performance-avoidance goal orientation ($B = 0.08, \beta = -.04, t = 0.51, p > .05$) variables failed to significantly increase in the explained variance of coding achievement, R^2 change = .001, F change (2, 188) = .13, $p > .05$.

In the final step, the inclusion of the last five variables (classroom mastery goal structure, classroom performance-approach goal structure, classroom performance-avoid goal structure, academic self-handicapping strategies, and cheating behavior) resulted in a nonsignificant increase of 2% in the explained variance of coding achievement, R^2 change = .02, F change (5, 183) = 1.59, $p > .05$. Among these variables, academic self-handicapping strategies was identified as the sole significant predictor of coding achievement, exhibiting a negative relationship ($B = -0.16, \beta = -.12, t = 0.51, p < .05$). However, CMGS ($B = 0.16, \beta = .11, t = 1.23, p > .05$), CPApGS ($B = 0.04, \beta = .01, t = 0.19, p > .05$), CPAvGS ($B = 0.07, \beta = .14, t = 0.49, p > .05$) and CB ($B = 0.01, \beta = .16, t = 0.08, p > .05$) found to be nonsignificant predictors of the academic achievement.

Table 4.13 Four-Step Hierarchical Multiple Regression Analysis Results

Predicted variables	B	$SE\ B$	β	t	p	sr	R^2 Change	R^2
Model 1							.068*	.068
(Constant)	53.403	2.179		24.507	.000			
School Location	-5.124	1.355	-.261	-3.781	.000	-.261		
Gender	-0.546	1.348	-.028	-0.405	.686	-.028		

Table 4.13 Four-Step Hierarchical Multiple Regression Analysis Results (cont'd)

Model 2							.477**	.545
(Constant)	31.517	6.064		5.198	.000			
School Location	-3.208	1.165	-.163	-2.753	.006	-.135		
Gender	-0.420	0.998	-.021	-0.421	.675	-.021		
MATS	0.504	0.104	.331	4.840	.000	.237		
RCTS	0.393	0.111	.235	3.533	.001	.173		
EL	-0.702	0.341	-.143	-2.057	.041	-.101		
GL	0.079	0.240	.024	0.328	.743	.016		
EFFI	0.095	0.089	.061	1.065	.288	.052		
ASTCE	0.060	0.021	.168	2.809	.005	.137		
Model 3							.001	.546
(Constant)	31.477	6.286		5.007	.000			
School Location	-3.212	1.171	-.163	-2.743	.007	-.135		
Gender	-0.416	1.003	-.021	-0.415	.679	-.020		
MATS	0.502	0.105	.330	4.792	.000	.236		
RCTS	0.392	0.112	.235	3.508	.001	.172		
EL	-0.714	0.345	-.146	-2.069	.040	-.102		
GL	0.076	0.241	.024	0.317	.752	.016		
EFFI	0.071	0.102	.045	0.694	.489	.034		
ASTCE	0.060	0.022	.167	2.764	.006	.136		
PAPGO	-0.022	0.122	-.012	-0.184	.855	-.009		
PAVGO	0.078	0.154	.035	0.505	.614	.025		
Model 4							.019	.565
(Constant)	34.803	6.859		5.074	.000			
School Location	-2.802	1.191	-.143	-2.352	.020	-.115		
Gender	-0.510	0.999	-.026	-0.511	.610	-.025		
MATS	0.498	0.106	.327	4.703	.000	.229		
RCTS	0.402	0.112	.241	3.584	.000	.175		
EL_av	-0.756	0.345	-.154	-2.192	.030	-.107		
GL_av	0.093	0.240	.029	0.386	.700	.019		
EFFI	-0.097	0.137	-.062	-0.707	.481	-.034		
ASTCE	0.048	0.022	.135	2.195	.029	.107		
PAPGO	-0.044	0.130	-.023	-0.336	.738	-.016		
PAVGO	0.059	0.156	.027	0.380	.705	.019		
CMGS	0.160	0.130	.108	1.233	.219	.060		
CPAPGS	0.039	0.208	.014	0.185	.854	.009		
CPAVGS	0.070	0.141	.036	0.492	.623	.024		
ASHS	-0.159	0.074	-.121	-2.145	.033	-.105		
CHEAT	0.012	0.157	.006	0.077	.939	.004		

Note. sr = semi partial correlation coefficient, * $p < .001$, ** $p < .001$.

Overall, the final model continued to significantly predict coding achievement and accounted for 57 percent of the variance in achievement, with an adjusted R^2 of .53 ($R^2 = .57$, $F(5, 183) = 15.82$, $p < .001$). Five of fifteen measures, which remained a robust predictor both in the second and third models and another variable added to the model in the fourth step statistically significantly contributed to the final model (Table 4.13). On the other hand, gender, GL, AE, PApGO, PAvGO, CMGS, CPApGS, CPAvGS, and CB were insignificant in predicting the coding achievement test scores of the fifth-grade student. When the unique relationship that each significant predictor has with coding achievement was examined, it was observed that mathematics achievement was the strongest predictor ($sr = .23$) followed by reading comprehension achievement ($sr = .18$), geographic school location ($sr = -.12$), extraneous load ($sr = -.11$), attitude towards coding education ($sr = .11$) and academic self-handicapping strategies ($sr = -.11$). To evaluate the generalizability of the model, the difference between R^2 and adjusted R^2 was calculated (Diff: $.565 - .529 = .036$) and found as 3.6%. This R^2 shrinkage indicated that if the model had been estimated using the entire population instead of a sample, about 3.6% less of the variance would account for in the outcome. Furthermore, considering the critiques on the R^2 value regarding its limitations in demonstrating the predictive capability of the regression model for a different dataset, to evaluate the cross-validation of the model, adjusted R^2 was calculated using Stein's formula and found as 0.49. As this calculated adjusted R^2 value did not differ substantially from the obtained R^2 value, the cross-validity of the model can be considered good.

4.2 Results of the Qualitative Data Analysis

4.2.1 Results of the Research Question 5

During the qualitative phase of the research, semi-structured interviews were conducted with a sample of 30 fifth-grade students from a total of 199 who had participated in the quantitative phase. These students were selected from each

participating school, with one student each representing low, medium, and high academic achievement in the ITS course from each class. A thematic analysis was conducted on student responses to gain deeper insights into their programming learning experiences. The analysis of the data revealed six main themes: cognitive demands, effective instructional approaches, collaborative learning approaches, independent learning approaches, goal setting, and affective aspects. Each theme is further categorized to capture specific aspects of the student experiences. The identified themes and their corresponding categories are presented in Figure 4.5.

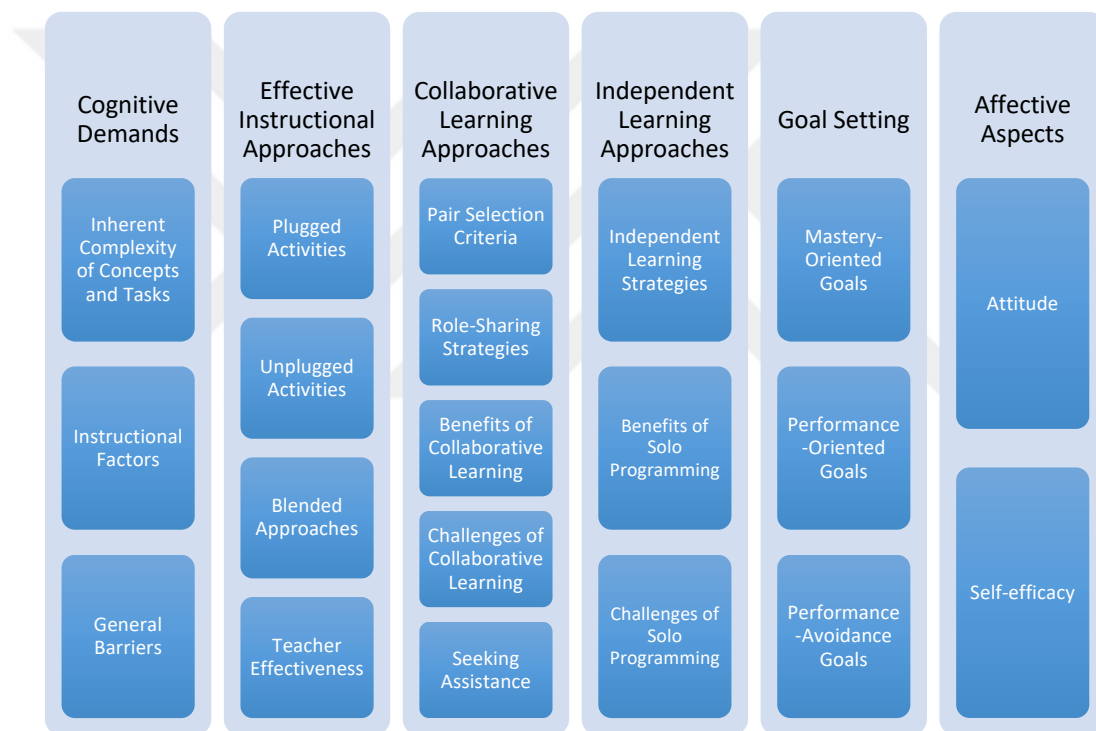


Figure 4.5 Themes and Their Corresponding Categories

4.2.1.1 Theme 1: Cognitive Demands

This theme explored the complexities that participants encountered while learning programming. The aim of this theme was to identify the factors contributing to their cognitive load by examining the inherent complexity of programming concepts and

tasks, the impact of instructional design, and the challenges posed by the learning environment. Table 4.14 provides a detailed overview of the categories under this theme, their corresponding codes, and the frequency of statements associated with each code.

Table 4.14 Distribution of Code Frequencies by the Theme of Cognitive Demands and Instructional Factors

Categories	Codes	<i>f</i>
Inherent Complexity of Concepts and Tasks	Diagramming programming logic difficulties	15
	Spatial reasoning challenges	14
	Managing iterative logic	6
	Limited code blocks challenges	3
	Comprehending code blocks functionality	3
	Integration of multiple concepts	2
	Sequencing and logical flow difficulties	1
Instructional Factors	Abstract concepts and confusing explanations	24
	Time constraints	8
	Unstructured learning	5
	Unclear task instructions	4
	Unsuitable scaffolding	1
Learning Environment Challenges	Access and equity issues	21
	Foreign language-related problems	4
	Login problems	3

4.2.1.1.1 Inherent Complexity of Programming Concepts and Tasks

This category examined the intrinsic aspects of programming concepts and tasks that significantly impact students' cognitive load. These intrinsic factors encompassed various challenges that learners face, influencing the mental effort required to grasp and apply programming concepts. The data revealed that participants frequently reported difficulties with programming logic, as evidenced by fifteen mentions of flowcharts as a particular challenge. Similarly, spatial reasoning posed a notable

obstacle for students learning to program, as highlighted in fourteen distinct instances within the data. Furthermore, the data indicated a significant challenge in managing iterative logic and loop structures, as evidenced by six mentions. Other challenges identified by students included problem-solving within constrained code blocks, understanding code block functionality, and difficulties with tasks requiring the integration of multiple concepts, algorithmic sequencing, and logical flow.

Diagramming Programming Logic Difficulties

Flowcharts are visual tools used to represent program logic. However, student feedback revealed difficulties in understanding which command to place inside which flowchart shapes, complicating their ability to effectively diagram coding logic ($f = 15$). This complexity was compounded by the need to grasp the abstract relationships between various shapes and their underlying concepts, such as processes and decisions. S27's confusion regarding "triangles" (referring to decision diamonds) exemplified this challenge. If students did not understand the purpose of decision diamonds and how to formulate questions or conditions within them, even a simple flowchart could become overwhelming.

[S27]: *In it, for example... When I started with a triangle, I didn't have any questions; I didn't know what to start with. I was struggling. Which command should I write and how should I write it?*

Additionally, four students expressed difficulty in identifying the specific actions or decisions represented by different flowchart symbols (e.g., decision diamonds and input/output boxes). These students struggled with remembering and distinguishing the meanings of different shapes in flowcharts, indicating challenges with concept retention and understanding. Understanding the abstract relationships between these shapes and their underlying concepts inherently demanded significant cognitive effort from the students. Moreover, student statements indicated that the way information was presented significantly affected the cognitive load they experienced. Poorly explained or inadequate practice in distinguishing different flowchart symbols led to memorization difficulties and confusion between shapes, which

added to the cognitive load as students struggled to recall the meaning of each symbol and how they connected to program logic.

[S27]: *I was confused about the questions. There were triangles, parallelograms, etc. I had some difficulty there. It was confusing. ...I didn't know what to put and which shape to put.*

Spatial Reasoning Challenges

Analysis of interviews revealed that many of the students struggled with tasks requiring spatial reasoning skills ($f = 14$). These skills, encompassing concepts like geometry (angles, degrees, and rotations) and distance calculations, are crucial for translating abstract geometric ideas into algorithms. Based on student feedback, integrating knowledge from geometry with programming skills was inherently complex and demanding. Participants' responses indicated challenges with tasks like navigating mazes requiring specific turns or manipulating shapes with precise rotations (as mentioned by S24). These tasks necessitated not only an understanding of programming concepts but also the application of geometric knowledge. Additionally, two participants (S15 and S26) expressed difficulty with directional commands ("left" and "right"). This issue was related to spatial orientation, which involved understanding and manipulating objects in relation to oneself. In programming tasks in the class, spatial orientation was essential for comprehending and using directional movement commands ("move forward," "turn left/right"). Student feedback indicated that those struggling with spatial orientation had difficulty visualizing how these commands translated to on-screen movements (as mentioned by S26).

[S19]: *I am not good at angles at all because I am not good at angles in math either. So, it affects my performance in the IT (Information Technology and Software) class as well.*

[S24]: *There are shapes there, for example, go 180. You will make it 120 or 145. I was undecided there, what should I do, 145, 120, 100? So, it was always necessary to try.*

Managing Iterative Logic

This code explores the challenges students encountered when working with nested loops, a fundamental programming concept that introduces multiple levels of iteration. Notably, the student tasks in this analysis specifically involved nested loops with only two iteration structures. Students struggled to understand the logic and structure of these constructs, particularly when determining which code blocks should repeat within nested loop structures ($f=6$). This was evident in the feedback from S30. The analysis of feedback from S22 also revealed difficulties in managing nested loop structures. Student reported challenges understanding the behavior of nested loops, particularly when the total number of iterations became large. This suggested a struggle with conceptualizing the interplay between nested loops and the resulting flow of control.

[S22]: *For example, when a character is there, it is a bit difficult for me to use that loop twice. For example, after repeating something three times, for example, when we put one more thing, a loop on the top layer, for example, when it was five times, something strange was happening. I couldn't really understand it.*

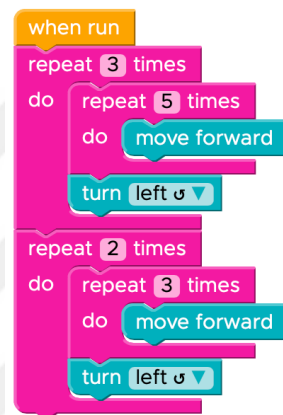
S6 expressed in the interview the challenges faced when applying nested loops in problem-solving tasks that involved navigating obstacles. These tasks required students to control a character and navigate through a complex environment filled with obstacles like ice patches and hazards (e.g., wildflowers). The inherent complexity lay in the need to manage multiple elements simultaneously, such as planning a route, executing movements, and adjusting strategies in real-time. This multitasking demanded strategic planning, spatial awareness, and a high level of problem-solving and decision-making skills. According to student comments, the

need to consider the location and behavior of multiple obstacles simultaneously increased the intrinsic cognitive load.

[S6]: *I had a hard time getting the zombie to the sunflower, it was difficult to get the zombie to the sunflower. Because, teacher, there are other flowers, wildflowers, you have to escape from them. Because there are broken ices, you are careless, you step on it, you fall anyway. Then the code is wasted.*



(a)



(b)

Figure 4.6 (a) Sample programming task on code.org (Course F-Lesson 8: Nested Loops in Maze/Level 10) about nested loops and (b) possible solution to this task

While game design elements like unnecessary distractions could add extraneous load, in this case, the ice patches and wildflowers appeared to be strategically placed. As illustrated in Figure X(a), these obstacles likely served a pedagogical purpose: to challenge students' understanding of nested loops and their ability to apply this concept in a practical setting. The complexity of navigating around these obstacles necessitated the use of nested loops to control the character's movement effectively.

Limited Code Blocks Challenges

This code captures feedback where S6 expressed that they struggled with tasks requiring the use of a limited number of code blocks, indicating challenges with coding efficiency and strategic problem-solving ($f = 3$). Code.org's puzzles often

challenged students with limited code blocks, including limitations on the total number of blocks and the number of times a specific block could be used. For example, maze navigation with limited moves puzzles that restricted the number of "move forward" blocks challenged students to carefully plan their route, considering the most efficient path and avoiding unnecessary moves. Similarly, loops and nested loops were also introduced as a way to solve problems with limited resources, requiring students to strategically plan code repetition. While constraints like limited block usage might seem challenging, they also aimed to encourage students to think creatively, optimize their solutions, and develop a deeper understanding of programming principles. This involved selecting the most efficient blocks and combining them in a way that achieves the desired outcome within constraints. Such tasks demanded higher-order cognitive skills such as analyzing, evaluating, and creating, as the student had to think critically about which blocks to use and how to use them effectively.

[S6]: *Code.org is actually great, but sometimes it makes you think that you should only be allowed to use it once. It's not difficult to set up normally, it's easy, but you really have to think about where to put it.*

Comprehending Code Blocks Functionality

Code blocks, fundamental building blocks in many programming environments, allow students to visually construct programs. Each code block has specific functionality and interaction patterns. However, challenges in understanding and using code blocks, particularly with variables, were mentioned in three different expressions. Students' statements revealed difficulties with grasping how variables work within code blocks. While they might understand the concept of variables theoretically (storing data), applying them practically in code blocks proved challenging. This finding suggested that difficulties with understanding how to connect code blocks involving variables, as well as assigning and manipulating values stored in variables within the code block structure, could be a contributing factor.

[S20]: *I don't understand how to use it.*

[S12]: *I had the most difficulty with the blocks we made in the last lesson because there were blocks I did not know. Because I don't know the codes. I barely learned how to use them.*

Integration of Multiple Concepts

Long-answer questions in programming often necessitate a deep understanding of complex material. They require students to integrate multiple concepts and synthesize information from various sources. The analysis of student feedback revealed a struggle with integrating multiple concepts within long-answer questions ($f = 2$). Answering these questions demanded not only recalling information but also applying, analyzing, and evaluating it. The emphasis on higher-order thinking skills also appeared to contribute to the intrinsic load.

[S16]: *For example, I had a lot of difficulty with the zombie, because I didn't know whether it would go to the right or to the left. It turns to the right. When there were too many blocks, I had a lot of difficulty. For example, I was doing it like this: I was turning to the bird's place, and I was trying to figure out which way it would go. Then, I was getting confused, and I was starting to slow down.*

Sequencing and Logical Flow Difficulties

Understanding the correct order of coding blocks requires a grasp of logical sequencing and control flow. S18 reflected in the interview that they struggled with maintaining the correct order of code blocks, often placing one block in the wrong position, indicating challenges with understanding the logical sequence and structure of code. This was complex because it involved understanding the cause-and-effect relationships between different parts of the code. The students had to understand what each block does and how they interact with each other to achieve the desired outcome. This foundational knowledge was critical to correctly sequencing the

blocks. Students also needed to develop the ability to debug their code by checking if the blocks were in the correct order and making adjustments as needed.

[S18]: ... *I'd mix up the codes. I'd accidentally substitute one for the other. It got mixed up.*

4.2.1.1.2 Instructional Factors

This category investigated the impact of instructional design on cognitive load. It explored how the design and delivery of instructional materials and activities contributed to unnecessary cognitive load for learners. The focus was on how poorly crafted instructional elements created confusion, hindered understanding, and ultimately impeded learning effectiveness. Key factors contributing to this issue included abstract concepts, confusing explanations, time pressures, lack of clear learning paths, unclear task instructions, and insufficient support for learners.

Abstract Concepts and Confusing Explanations

This code addressed the challenges in understanding abstract programming concepts due to inconsistent or misleading explanations. Variables, fundamental building blocks in programming, represent concepts that can store and manipulate data. However, grasping their purpose and functionality could be challenging for students, as evidenced by the difficulties being emphasized 24 times by participants in this study. Two of the students (S26 and S21) expressed that they struggled with understanding and working with variables because the examples or explanations provided seemed inconsistent or misleading. For instance, S26's confusion about "five fingers changing" highlighted how real-world analogies could be misinterpreted, leading to a distorted understanding of variable behavior.

[S26]: *For example, the teacher tells me that five fingers never change, but when I do it like this, it becomes ten fingers, I think it changes. There was a little bit of a discussion there. After that, I started to have a lot of trouble. I was a little surprised there.*

[S21]: *I mean, it always seems to me... It is variable, for example, it seems like they can all change. But this depends on the thing, that is, it depends on the puzzle there. If the puzzle is like that, it is constant. But it can also be variable. ... The basis of the constant actually depends on the thing; this is another strange idea of mine; the basis of the constant depends on the program there. Whatever it is set up, that's the constant. But if we look at it, everything can change. That makes it difficult for me to understand.*

Time Constraints

Student feedback highlighted the issue of insufficient class time for effective learning, particularly for complex topics like nested loops and variables ($f = 8$). Students expressed a desire for more frequent or longer class sessions to allow for deeper understanding and practice. Restricted class time could hinder students' ability to grasp complex concepts thoroughly. In the case of nested loops and variables, students mentioned encountering difficulties and having areas where they still need improvement. This suggested that the current class duration is inadequate for providing sufficient exposure and practice with these challenging topics.

[S3]: *More lessons per week... Two lessons per week is not enough. A topic can be emphasized more. For example, when we moved on to nested loops, there were points where I encountered problems that I sometimes could not solve in my head. Or in variables... For example, there were places in variables that I still could not do. The subject could have been emphasized more. It would be better if there were more lessons per week.*

Unstructured Learning

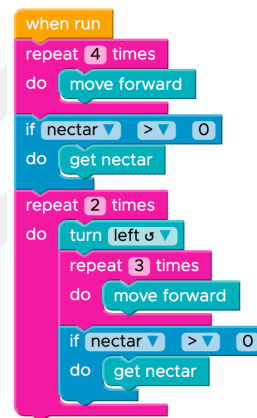
Five of the students' expressions showed difficulty with the puzzle set, exemplified by the screenshot of a sample level in Figure 4.7. This puzzle set, which involved tasks requiring an understanding and application of various programming concepts such as conditional statements, loops, and variables, could be challenging for students new to programming (as explained under the code of 'integration of multiple

concepts’). However, these students were introduced to conditional statements before learning about variables. As a result, they might have struggled to understand the conditions for moving the bee (e.g., 'if there is nectar') without knowing how to store and manipulate data related to nectar availability using variables.

[S9]: *We didn't know how much nectar there was. It was very difficult; if there was nectar, we had to take the nectar and move forward. We had difficulty with that. I mean, it was like this, there was one square, there was nectar all around. I was confused about whether to move the bee forward or take the nectar.*



(a)



(b)

Figure 4.7 (a) Sample programming task on code.org (Course 2 – Lesson 13: Bee Conditionals/Level 8) about conditional statements and (b) its solution

Unclear Task Instructions

Clear and concise instructions are crucial for minimizing cognitive load and maximizing learning in coding education. This is especially important for unplugged activities, which involve using physical objects to represent programming concepts. Four students’ statements indicated that they experienced significant difficulty when interacting with unplugged objects due to the unclear instructions. In these cases, unclear instructions for using physical objects like cups to represent direction changes led to confusion because students were unsure how to manipulate these

physical objects to practice programming logic and required students to exert extra mental effort to interpret meaning, fill in gaps, and understand the task requirements. This might increase extraneous cognitive load, diverting resources away from the core learning objectives of the unplugged activity.

[S17]: *And those non-digital, non-computerized ones confused me. That glass confused me so much that my brain burned.*

Unsuitable Scaffolding

Student feedback underscored the critical importance of appropriate pacing and scaffolding in programming education. S3 detailed their initial ease with learning basic loops on code.org but expressed significant frustration with the sudden introduction of nested loops. This sudden transition from simple to complex examples overwhelmed students, thereby hindering their effective learning. When students were confronted with challenging concepts such as nested loops without adequate preparatory instruction, they might have experienced cognitive overload. This overload might have occurred due to the necessity of processing excessive information simultaneously, which complicated the comprehension of underlying concepts.

[S3]: *In nested loops, it was like this. We were playing very simple when we were training on code.org at the beginning: Go 4 steps forward or... At first, we learned repeated loops. It was easier to repeat instead of writing too much. It wasn't too hard to learn. But the nested loops suddenly became difficult.*

4.2.1.1.3 Learning Environment Challenges

The learning environment can significantly impact student success. This category explored three key challenges identified through student feedback: access and equity issues, foreign language-related problems and login problems.

Access and Equity

The study highlighted concerns regarding access and equity in technology resources ($f=21$). Several students pointed to difficulties caused by outdated or malfunctioning computers, which hindered their participation in essential educational learning environments like Code.org. This issue particularly disadvantaged students lacking reliable equipment, especially in suburban schools. S9 specifically mentioned the added struggle of using a malfunctioning computer to access Code.org, a crucial tool for their coding education. Furthermore, the lack of technology at home further restricted learning opportunities, as evidenced by eight students who expressed difficulty engaging in coding activities due to the absence of essential devices, such as computers or tablets. These concerns underscored the potential for unequal learning experiences and the critical importance of addressing the digital divide.

[S13]: *Also, computers break down a lot, I would like to change them.*

[S9]: *I didn't work on coding because... I didn't work on it. I don't have a computer.*

Foreign language-related problems

Participants' statements pointed to problems with Turkish language support within the digital programming learning environment during their interviews ($f = 4$). Students S5 and S26 reported that despite selecting the Turkish language option, some parts of the platform remained in English. These students expressed difficulty in understanding instructions, introductory videos, and technical terms that were not in their native language. This language barrier might have added an extra layer of cognitive load, hindering their ability to grasp complex concepts and follow instructions effectively. This issue underscored the importance of comprehensive language support in educational programming environments to ensure that non-native English speakers could fully engage with the learning material.

[S5]: *For example, in Course F, around lesson thirteen, even though I set the language to Turkish, we still have to speak in English there.*

[S22]: *Well, you know the videos at the bottom, it would be better if they were translated into Turkish.*

Login issues

The coding learning environment offered user-friendly login options for students. Students could access the platform using a section code provided by their teacher, followed by their name and a text-based password or picture password. The picture passwords were printed out and distributed to students as a physical reminder. Nevertheless, two students expressed challenges related to logging into the coding website. For instance, S19 highlighted the frustration of forgetting passwords and the inability to access the learning environment without them. This forgotten password obstacle could add to the cognitive load and disrupt the learning flow for students.

[S19]: *How can I say? It seemed a bit difficult to have a password. I mean, when we forget our paper and password, we may not be able to enter without a password.*

4.2.1.2 Theme 2: Effective Instructional Approaches

This theme addressed the design and implementation of teaching strategies that promote participating students' learning and engagement in programming education. It encompassed four categories that reflected diverse methodologies IT teachers employed to maximize student engagement and learning outcomes. The categories included plugged activities, unplugged activities, blended approaches, and teacher effectiveness. The codes under each category and the number of participants stated in the relevant code are given in Table 4.15.

Table 4.15 Distribution of Code Frequencies by the Theme of Effective Instructional Approaches

Categories	Codes	<i>f</i>
Plugged Activities	Facilitated learning	64
	Rich content	14
	Permanent learning	13
	Learning by doing	5
	Debugging tasks	4
	Opportunities for revision and mastery	3
Unplugged Activities	Introduction and orientation	19
	Active engagement	5
	Real-world relevance	5
Blended Approaches	Blending traditional and digital methods	4
Teacher Effectiveness	Clear and effective explanations	23
	Supportiveness	9

4.2.1.2.1 Plugged Activities

Plugged activities in this context referred to the direct use of computers and programming software, enabling students to practice and apply coding concepts within a digital environment. In the current study, the coding learning environment Code.org served as the primary tool for all plugged activities.

Facilitated Learning

A substantial number of participants in the study expressed a preference for engaging in technology-mediated activities, particularly those offered by Code.org, for the acquisition of programming concepts ($f = 64$). These participants indicated that such activities facilitated their comprehension and application of coding skills more effectively than traditional, non-technology-based methods. One participant, S12, highlighted the positive impact of Code.org's puzzles on their learning experience. Furthermore, four participants (S15, S17, S19, and S25) directly compared technology-mediated activities with non-technology-based methods and found the former more effective in fostering an understanding of programming skills. This preference for interacting directly with code blocks on a computer

aligned with the participants' perceptions that technology-mediated activities offered a more advantageous approach to learning programming, as explained by S17. The feedback was particularly positive towards Code.org, with S18 and S21 appreciating its clear explanations. Besides that, S9 remarked that the block-based structure of the coding learning environment simplified the development of codes, making it easier to piece together solutions. In conclusion, the experiences of the participants provided strong evidence supporting the efficacy of technology-mediated activities, particularly those offered by Code.org, in facilitating a more successful learning experience for programming education.

[S9]: It facilitated my learning and made a significant contribution. By doing it this way, I got used to it and started to do it very quickly. I improved myself in coding; code.org was helpful to me. Because I solve it by piecing parts together, it becomes easier. This way, I was learning coding better.

Rich Content

Fourteen instances of feedback commended the diverse array of activities offered by Code.org, highlighting its capacity to accommodate various learning styles. The structured learning approach provided by the learning environment was particularly valued, as it facilitated a clear, progressive path that not only challenged the students but also fostered a sense of mastery and bolstered their confidence, as noted by S3. The rich content available on Code.org was instrumental in maintaining student engagement (S17, S18, and S29), while its well-conceived activities significantly deepened their understanding of programming concepts (S3). Furthermore, students underscored the benefit of Code.org's detailed explanations, which they found superior to those on other platforms they had experienced. Overall, the positive feedback on Code.org highlighted its effectiveness in creating a rich and engaging learning environment that promoted deeper understanding and skill development in programming.

[S3]: *I think I learned more in detail thanks to Code.org, in some topics. For example, I recently learned about variables and really struggled with them. It was very instructive for me in this regard. As I said, I already knew most of the terms in Scratch, or most of the block terms. However, I think I learned better about nested loops and variables. Variables were also covered in Scratch, but not in such detail. I definitely think I learned variables in more detail. It was really good.*

Permanent Learning

The study revealed a strong connection between hands-on learning and knowledge retention in programming ($f = 13$). Students noted that the activities provided by Code.org played a crucial role in enhancing their long-term understanding of the subject (S14, S15, S21, S25, and S30), demonstrating the coding platform's effectiveness in promoting deeper and more enduring learning. Additionally, several students (S5, S21, S27, and S28) expressed a desire for increased hands-on practice, believing it would improve their learning outcomes and reinforce knowledge retention. They emphasized that active engagement with the material, rather than passive listening in a classroom setting, leads to better comprehension and memory retention (S21 and S30). This focus on permanent learning underscored the importance of incorporating active learning strategies into programming education to facilitate a more profound and lasting understanding of coding principles.

[S21]: *I think it becomes more permanent. Because the explanation only goes so far... In the classroom environment, it's already difficult to understand and define things well. So, doing it here makes it more permanent.*

Learning by Doing

Learning by doing emerged as a powerful learning strategy in fifth-grade students' experiences within programming education ($f = 5$). Students highlighted the effectiveness of hands-on engagement and direct interaction with coding tasks in their learning. For instance, S16 contrasted the difficulty of understanding concepts

explained by a teacher explanation on a board with the ease of using Code.org. This emphasized how hands-on activities made complex concepts more accessible. S20 discussed how the practical application of coding steps facilitated the internalization of knowledge. Similarly, S30 emphasized the importance of the ability to "see and do" in understanding the application of coding concepts. Moreover, he expressed a preference for self-directed learning through hands-on activities, highlighting the value of active engagement in promoting a deeper understanding. Collectively, these student experiences strongly supported learning by doing as an effective approach for facilitating a more meaningful and successful learning experience in programming education.

[S16]: *It had a lot of impact because if a teacher had explained it by drawing on the board, I wouldn't have understood it at all. But Code.org was easier for me. If it weren't for that, and the teacher had explained it by drawing, I would have understood a little, but not much.*

[S30]: *But I think it is of better quality when we do it ourselves.*

Debugging Tasks

Results from the study identified debugging tasks, where students correct partially completed or erroneous code, as an effective learning tool in programming education, based on four statements from two participants. Results from the study indicated that debugging tasks, which enabled students to focus on particular parts of the code rather than constructing the complete program, reduced the mental effort required. For instance, one student (S9) mentioned that working with a partially completed solution facilitated a more focused approach to particular code aspects, making it easier to complete tasks. Another student (S21) valued the structured and guided nature of debugging tasks, which helped them concentrate on understanding and applying specific programming concepts more effectively. Overall, debugging tasks offered a valuable strategic method to enhance programming education for children by reducing cognitive load, promoting focused practice, and fostering a deeper understanding of programming fundamentals.

[S9]: *For example, code.org would create the tasks and ask me for the angles and such. That made my job easier. It was an advantage. Sometimes it was easier. It would combine the parts, and I would set the angles, like ninety degrees...*

Opportunity for Revision and Mastery

The study revealed that the computer-based learning environment afforded valuable opportunities for students to revise their work and achieve mastery ($f = 3$). One student (S19) recognized the immediate feedback provided by the computer, which enabled the prompt identification and correction of errors. This real-time feedback mechanism facilitated a more effective understanding of concepts and fostered a sense of self-correction. Moreover, the opportunity to attempt tasks multiple times, as highlighted by S19, was deemed advantageous. This iterative process allowed students to refine their code, experiment with different approaches, and enhance their understanding. Overall, the environment promoted a growth mindset by encouraging students to learn from their mistakes and persevere through challenges. While another student (S21) acknowledged the broader value of exploring diverse perspectives and new approaches, the primary benefit highlighted by the students was the opportunity to revise and perfect their work, leading to a deeper understanding of programming fundamentals. This emphasis on revision and mastery underscored the critical role of adaptive learning technologies in fostering educational advancement in programming disciplines.

[S19]: *Because on the computer, we can see our mistakes immediately. But on paper, we can't see our mistakes. As I said, on the computer, we can see our mistakes and correct them accordingly. If there are correct ones, we review them again, as they might be wrong.*

4.2.1.2.2 Unplugged Activities

This category investigated the role of unplugged activities in enhancing programming education. Through an in-depth analysis of student perspectives, how these computer-free experiences contributed to overall learning outcomes was assessed. The findings emphasized the benefits of unplugged activities in creating a dynamic and effective learning environment. The analysis focused on several key aspects, including introduction and orientation, active engagement, and real-world relevance.

Introduction and Orientation

Instructor-led introductions were identified as critical for student success in programming. Nineteen instances emphasized the value of pre-laboratory lectures and demonstrations in establishing a strong foundation for hands-on practice. These introductions provided clear explanations of new concepts (S18, S19, S20, S22, and S24) and demonstrations that facilitated understanding (S22 and S23). Such initial orientation was efficient in helping students approach practical exercises with a clearer grasp of expectations and procedures (S24). While some students found unplugged activities beneficial (S27), the majority highlighted the effectiveness of the instructor's explanations in preparing them for successful computer-based learning (S2, S3, S4, S6, and S8). Overall, the importance of well-designed introductory sessions in programming education was strongly underscored by the experiences shared by students in this study.

[S19]: *First, he showed us on the smart board during the first hour. We started doing it on the smart board. He began correcting our mistakes. Then we tried to do it on our own on the computers. So, as I said, first, he teaches us on the smart board and explains it. Then we do it. It would be nice if it continued like this; we would like to keep it this way.*

Active Engagement

The qualitative analysis underscored that active engagement in unplugged activities enhanced students' learning experiences by reducing distractions and fostering a more focused learning environment ($f = 5$). Participants (S6, S21, and S22) reported that unplugged activities allowed them to concentrate intently on the tasks at hand without being distracted by extraneous elements, thus maximizing their cognitive resources for processing relevant information. For example, one student (S6) noted that classroom activities that involved physical movement and hands-on manipulation of objects were more instructive compared to computer-based tasks, as they enhanced engagement and learning. Another student (S21) highlighted the effectiveness of being called to the board to solve problems, describing it as a more engaging and interactive learning experience that promoted deeper understanding. Furthermore, S18 emphasized that directly applying concepts themselves led to better comprehension and skill acquisition. These findings indicated that unplugged activities, which involved active participation and physical manipulation, not only improved focus but also significantly enhanced students' engagement and understanding. This approach provided a valuable contrast to digital methods, offering a dynamic and interactive learning atmosphere that could lead to more effective education outcomes.

[S6]: *Teacher, I think that what we did in the classroom was more instructive. Because on the computer, you only move the mouse and the thing. But in the classroom, you move yourself, you adjust things yourself.*

Real-World Relevance

The importance of real-world connections emerged in the study (S23, S26, S29, and S8), particularly when discussing unplugged activities ($f = 5$). Students indicated that these activities were essential for demonstrating the practical utility of programming knowledge in everyday contexts beyond the confines of the classroom. Students believed that this ability to link programming concepts to real-life scenarios made the learning material more relatable and meaningful, thereby fostering a deeper

understanding of the reasons behind their learning efforts (S8). Witnessing the practical applications further enhanced student motivation as they recognized the relevance of programming skills in their personal lives (S23, S26, S29, and S8). By connecting the theoretical knowledge with practical real-world applications, unplugged activities provided an engaging and pertinent learning experience that significantly improved learning outcomes.

[S8]: *I learned that we can use coding, that is, commands, in real life as well.*

4.2.1.2.3 Blended Approaches

This category illuminated the advantages of blended approaches, demonstrating how the integration of traditional and digital methods facilitated improved learning outcomes and enhanced knowledge retention.

Blending Traditional and Digital Methods

While most students expressed enjoyment of computer-based activities and the use of Code.org, a group of participants (S13, S16, and S21) emphasized the importance of blending traditional, read-write methods with digital activities ($f = 4$). They advocated that this hybrid approach, within a technology-enhanced learning environment, enhanced learning outcomes and knowledge retention. These students particularly noted the advantages of transcribing their computer work onto paper, stating that this practice reinforced their understanding and augmented their ability to review and retain information. One student (S13) detailed that writing things down facilitates a deeper engagement with the material, contrasting with the transient nature of digital interactions where information can be easily accessed but quickly forgotten. The physical act of writing created a durable record that encourages students to revisit and consolidate their understanding of the concepts. This student proposed a system where learners would document their digital work on paper, effectively creating a personalized study guide for continual reference. This approach underscored the value of a blended learning environment that leveraged

the strengths of both traditional and digital educational practices to foster a more comprehensive and enduring learning experience.

[S13]: *Because, teacher, we are both writing and reading. I mean, we write with our own hands and we read. ...On the computer, for example, you press a key, but you can't do it completely. Writing down what we did on paper... Because, teacher, when you show it on paper, you read it, you read it a second time, and then you can put it in a folder, and if you forget it, you can look there and do it. On the computer, you might not be able to access it; it might not be saved, and it could be lost.*

4.2.1.2.4 Teacher Effectiveness

The impact of teacher effectiveness on student learning experiences was examined under this category. Based on the findings, this examination focused on two key aspects: the clarity and effectiveness of explanations provided by the teacher and the level of support and assistance offered to students.

Clear and Effective Explanations

The qualitative analysis underscored the critical role of effective teaching methods and teacher performance in enhancing students' learning experiences. The ability of the IT to communicate subject matter effectively was frequently emphasized in student responses ($f = 23$). These students expressed high levels of satisfaction with their teacher's performance, noting that clear and effective explanations significantly enhanced their understanding of programming concepts. For instance, S26 directly commended the teacher's lucid explanations, and S15 observed a noticeable increase in knowledge acquisition from the lessons. S16 specifically appreciated the teacher's skill in simplifying complex topics, a sentiment echoed by others who valued the clarity and efficacy of the instruction (S20 and S21). These positive evaluations of the teacher's effectiveness underscored the vital role of clear and comprehensive instruction in improving students' educational experiences and outcomes. Students'

statements showed that the teacher's ability to effectively communicate complex coding concepts helped students overcome challenges.

[S26]: *My teacher teaches coding very well. His explanation is very effective.*

Supportiveness

The study also highlighted the importance of teacher supportiveness in facilitating student learning ($f = 9$). Participating students admired the IT teacher's readiness to assist students encountering difficulties and to elaborate on concepts as needed. This illustrated the teacher's proactive approach and effectiveness in addressing student needs in real-time. Students mentioned the teacher's attentiveness to the entire class despite its size, their ability to get individual help when needed (S24 and S26), and their readiness to explain concepts in greater detail upon request (S26). Overall, the teacher's supportive approach, evident in their attentiveness to student needs and willingness to provide assistance, contributed positively to the learning environment and fostered a more inclusive and responsive educational experience.

[S24]: *The class is crowded, with 28 people. The teacher attends to all 28 of us. For example, when I say that I get frustrated when I can't do something, at those times, I call the teacher. The teacher explains it to me, and then I can easily get past that part. It really helps a lot.*

4.2.1.3 Theme 3: Collaborative Learning Approaches

This theme investigated the social cognitive factors influencing students' programming experiences and the application of collaborative learning strategies within the educational context. It examines collaborative learning in programming education through the following categories: pair selection criteria, role-sharing strategies, benefits of collaborative learning, challenges of collaborative learning, and seeking assistance. The categories and sub-categories, along with their corresponding codes and the frequency with which each issue was reported, are presented in Table 4.16.

Table 4.16 Distribution of Code Frequencies by the Theme of Collaborative Learning Approaches

Categories/Sub-categories	Codes	<i>f</i>
Pair Selection Criteria	Social compatibility	15
	Skill and expertise	2
Role-Sharing Strategies	Imbalanced turn-taking	27
	Regular turn-taking	26
Benefits of Collaborative Learning	Mutual learning and knowledge sharing	42
	General positive perceptions	34
	Enhanced problem-solving	22
	Shared responsibility	7
Challenges of Collaborative Learning	Unequal participation	14
	Conflicts over resource sharing	8
	Reduced engagement	7
	Conflicts over problem-solving approaches	7
Seeking Assistance		
Source of Assistance	Teachers as a source of support	32
	Peers as a source of support	14
Reasons for seeking assistance from peers	Teacher unavailability	3
	Familiarity	2
Reasons for seeking assistance from the teacher	Clear explanations and guidance	3
	Teacher expertise	2
	Self-perceived proficiency	1
Peer Support Strategies	Unproductive collaboration strategies	16
	Constructive collaboration	9

4.2.1.3.1 Pair Selection Criteria

Feedback from students engaged in pair programming suggested a primarily student-driven approach, with some instances of teacher-assigned pairings. The analysis revealed that students frequently opted for self-selection, basing their choices on specific criteria. This category examined students' preferences for pair programming and the factors they considered when selecting partners or forming collaborative groups.

Social Compatibility

While the analysis revealed self-selection as the predominant approach for pairing, a deeper examination explored the criteria students employed when choosing partners. According to the results, when forming pairs for programming tasks, a significant number of students prioritized social compatibility ($f = 15$). They gravitated towards familiar faces and friends, valuing the comfort and ease of working with someone they knew. This preference for friendly partners, as evidenced by student feedback, stemmed from the creation of a more supportive learning environment. Students indicated that existing friendships or prior acquaintances fostered trust and open communication, which are crucial for effective collaboration.

[S19]: *Because I am better with that friend. I mean, our houses are next to each other. I have a better relationship with him. We have a better friendship with him. Firstly, I used to work with another friend of mine. Then our friendship ended, and we had a fight. So, we asked permission from the teacher, and I started working with him.*

Skill and Expertise

Notably, the emphasis on selecting a partner with strong computer science skills and knowledge was highlighted only twice in the responses of the interviewed participants. S17 emphasized the importance of perceived technical expertise. This perspective underscored the potential advantages of complementary skill sets in pair programming. The general lack of focus on these complementary skills suggested that students might not fully appreciate the benefits such an approach could offer in enhancing collaborative work.

[S17]: *He is a computer expert. He can't do the easy stuff. I chose him because he is a smart kid. Intelligent.*

4.2.1.3.2 Role-Sharing Strategies

This category explored the methods and approaches used to assign and manage roles among students during pair programming activities. It specifically examined whether the role distribution led to balanced or imbalanced turn-taking among the participants.

Imbalanced Turn-Taking

Fifteen of the students indicated that they adopted imbalanced turn-taking strategies during pair programming activities ($f = 27$). Imbalanced turn-taking occurs when students choose primary roles as either navigator or driver, predominantly maintaining them throughout the activities. Statements revealed that when students opted for the driver role, controlling the keyboard and mouse, their partner remained seated beside them as the navigator, offering verbal assistance in solving programming tasks. This arrangement resulted in an imbalance in role distribution. Another example involved dividing keyboard and mouse responsibilities (e.g., S14). While this approach appeared more balanced, the student controlling the mouse ultimately played a more dominant role, assuming the driver position during drag-and-drop puzzles.

[S14]: *He usually used the keyboard, I used the mouse. It usually continued like this.*

Regular Turn-Taking

Regularly taking turns emerged as a key feature in many students' pair programming sessions and was noted with a frequency comparable to that of strategies addressing imbalanced turn-taking. This balanced approach ensured that both partners actively participated and contributed. Students such as S10 and S11 highlighted the importance of equitable participation in programming tasks. Taking turns also involved rotating tasks, as seen in S16's explanation of switching between playing different levels. Similarly, S19 described alternating between keyboard and mouse

duties. This balanced participation facilitated shared ownership of the problem-solving process and enhanced collaborative learning.

[S16]: *We usually took turns playing with my friend. In the levels from one to nine, I would play one, then three. So, I play one, they play two, I play three, they play four.*

4.2.1.3.3 Benefits of Collaborative Learning

Pair programming, a collaborative learning approach, had demonstrably positive effects on student learning outcomes, as evidenced by the interview data. This category highlighted how collaboration could enhance the programming learning experience for students. Key benefits of collaborative learning identified by students included mutual learning and knowledge sharing, general positive perceptions, enhanced problem-solving and shared responsibility.

Mutual Learning and Knowledge Sharing

Qualitative data revealed that a collaborative learning environment fostered a powerful dynamic of mutual learning and knowledge sharing ($f = 42$). Interviewed students highlighted the value of exchanging ideas, learning from each other's strengths, and correcting mistakes together. This collaborative approach led to a deeper understanding of concepts and enhanced problem-solving skills. At the core of this benefit lay reciprocal learning, as evidenced by student expressions. Students acted as both teachers and learners, sharing their knowledge and perspectives during programming tasks (S1, S14, S15, and S17). Students noted that this exchange broadened their understanding of the subject matter and exposed them to different viewpoints. Additionally, the study's findings demonstrated that students benefited from each other's strengths. Stronger partners guided their peers, while those struggling received valuable support, as evidenced by the statements of S12, S16, and S26. The findings indicated that collaborative problem-solving facilitated error correction. According to the statements of S16 and S19, by explaining their thought

processes, students were able to identify and rectify mistakes, leading to a more refined understanding. Finally, mutual learning fostered a deeper understanding of concepts as students discussed and explained ideas to each other, a process highlighted by S21 and S24. This approach reinforced comprehension and solidified knowledge. Examples such as S1 explaining concepts to a peer or S17 learning coding basics from a friend illustrated the power of this interaction. Overall, collaborative learning fostered a supportive environment where students were able to learn from each other's strengths, overcome challenges collectively, and achieve a deeper understanding of the subject matter.

[S16]: *I think he learned, but... I mean, for example, when we first started, he couldn't understand right and left very well. I explained that to him. I was confusing things, you know, I was confusing things like repeating this thing five times. He taught me that too.*

[S26]: *In general, I taught him a lot on code.org, but beyond that, he taught me a lot about the basics of computing.*

General Positive Perceptions

The research findings indicated that there were 34 statements reflecting students' general positive views on collaborative programming learning. Eight students (S2, S4, S10, S11, S14, S19, S22, and S25) reported no drawbacks in pair programming, especially when working with compatible partners. Fourteen students indicated a preference for pair programming. This suggested that many students recognized the benefits of teamwork, communication, and knowledge sharing that pair programming facilitates.

[S14]: *Given the choice, I would sit with my friend again.*

Enhanced Problem-Solving

Feedback from students highlighted the significant benefits of collaborative learning in enhancing problem-solving skills ($f=22$). Students reported that working together with peers facilitated a better understanding and quicker solutions to programming

challenges. They emphasized the value of diverse perspectives, immediate assistance, and the combined intellectual effort that comes with teamwork. For instance, S13 mentioned that working in pairs allowed them to ask a friend for help instead of relying solely on hints, which facilitated the learning process. S14 and S15 highlighted the ease of getting assistance from friends when stuck on a problem, while S19 noted the mutual correction of mistakes. S21 also valued the multiple perspectives that come from collaborative work, explaining that different viewpoints helped in understanding and solving problems more effectively. One student (S30) pointed out that collaboration often led to faster problem resolution. Additionally, S6 referred to the proverb "unity is strength," illustrating the belief that collaboration enhances problem-solving capabilities. S7 noted that solving problems together was easier because they could leverage each other's knowledge. Overall, these insights illustrated that peer collaboration not only enhanced problem-solving efficiency but also fostered a supportive learning environment where students could share ideas and overcome challenges together.

[S7]: *Because it is easier. It is easier because we both solve it. Both of us can see what we cannot do; for example, one of us knows, and one of us does not.*

[S21]: *... for example, your friend looks at something from one perspective, you say, let's look at it from this perspective, that is, a multiple perspective. He says it is necessary to proceed from this logic, for example, you are doing a different logic. It is a different point of view. In the questions you cannot solve, you need to change your perspective on the problem.*

Shared Responsibility

Shared responsibility emerged as a significant benefit of collaborative learning, particularly during pair programming, as highlighted by the students ($f = 7$). Based on student statements, the research findings demonstrated that this shared approach allowed students to take breaks and avoid burnout. When one student felt tired, the other could take over, keeping both pairs refreshed and engaged. S6 noted that the collaborative approach not only helped manage fatigue by distributing the workload

but also contributed to maintaining engagement by keeping both partners actively involved in the problem-solving process. It also facilitated an efficient division of labor, allowing students to leverage their strengths and interests to deal with tasks, leading to quicker and more effective task completion. Overall, students found shared responsibility to be a valuable asset in collaborative learning, making the experience both enjoyable and highly productive.

[S4]: *It is more enjoyable. Instead of working alone, you alternate with your partner. While he is doing it, you are watching him, while you are doing it, he is watching you, it can also happen. Also, from time to time, your friend also rests after writing or using, you also rest, it is nice.*

4.2.1.3.4 Challenges of Collaborative Learning

Analysis results showed that while collaborative learning provided numerous benefits, it also presented several challenges for the participating students. This category explored key obstacles identified in student interview data, providing insights into potential areas for improvement. These challenges included unequal participation, conflicts over resource sharing, reduced engagement, and conflicts over problem-solving approaches.

Unequal Participation

Several students expressed concerns about not having equal opportunities to participate and contribute due to imbalanced turn-taking strategies ($f = 14$). Student experiences highlighted various forms of imbalance participation. In some instances, students intentionally adopted imbalanced turn-taking strategies, with each student selecting a primary role (navigator or driver) and maintaining that role throughout the session (e.g., S13). However, there were also instances where one partner dominated the activity, taking control of the computer and leading the task without adequate involvement from their partner, leaving the other feeling passive and unable to contribute meaningfully. According to the students' expressions, this not only

hindered the learning experience of the less active participant but also created a sense of unfairness. Additionally, S3's statement highlighted that inefficient task distribution can result in one partner feeling overwhelmed while the other remains underutilized, thereby disrupting the collaborative flow. Furthermore, as S29 stated, when one student dominated, the other had limited opportunities to practice their own skills, potentially hindering their individual growth. These insights highlighted the challenges of ensuring equal participation in collaborative learning environments. The findings revealed that uneven engagement led to one student dominating the task while the other became passive, which negatively impacted the learning experience and outcomes for both.

[S17]: *Because he does not give me control of the computer, he no longer does. I rest (laughing). I lean back like this again. I take examples from what he does. Sometimes, he does things so that I can do them; sometimes, he allows me to do them.*

Conflicts Over Resource Sharing

Eight statements from the data highlighted conflicts over resource sharing, particularly regarding the use of the computer and other equipment, as a significant challenge in collaborative programming. One student (S12) explicitly mentioned disliking the need to share a computer with a partner, expressing frustration with collaboration and resource limitations. Similarly, S13 and S17 described frequent arguments over who would control the keyboard and mouse, noting that sharing a computer often led to frustration, especially when students had to wait for their turns. The findings indicated that this frustration could hinder students' engagement and motivation in the learning process. S23 pointed out that their partner always wanted to use the keyboard, leading to constant conflicts. Additionally, students expressed a preference for solo programming to avoid conflicts arising from shared resources, indicating that some students prioritize individual work environments for a smoother learning experience. These insights highlighted the need for better resource management strategies to enhance the effectiveness of collaborative learning.

[S12]: *My least favorite thing is that I have problems sharing the computer with my friend with whom I share the computer.*

Reduced Engagement

The analysis of student data revealed another challenge: reduced engagement during their partner's problem-solving phase ($f = 7$). The data indicated that while pair programming was designed to foster collaboration and teamwork, its effectiveness was diminished when passive participation occurred. Four students (S1, S16, S24, S8) mentioned not actively participating while their partners were working on problems. This included not following along with the partner's thought process or code implementation unless they directly asked for help and offering little feedback or suggestions during their partner's lead. Limited participation was evident as students provided minimal verbal or coding contributions during their partner's lead. S24 highlighted that taking turns without explaining their processes to each other did not benefit either partner. Similarly, S8 noted that merely alternating the use of the mouse and keyboard did not contribute to their learning. Research results demonstrated that students who were not actively engaged in problem-solving but only observed the process missed the chance to interact deeply with the material and fully comprehend the concepts being taught.

[S16]: *Sometimes, you know, I was staying while he was doing it, I wasn't looking at him. I helped him when he asked for help, but I usually did not look at the questions he did.*

Conflicts Over Problem-Solving Approaches

Analysis results indicated that while pair programming effectively fostered teamwork and problem-solving skills, it was undermined by disagreements on problem-solving approaches ($f = 7$). Although some disagreements could be productive, significant conflicts, as reported by four students, could impede collaboration and communication, thereby diminishing the effectiveness of the pair programming experience. For instance, S21 described situations where neither

partner was willing to compromise on their approach, resulting in persistent disagreements. Such conflicts could lead to missed learning opportunities, as highlighted by S3, who mentioned disagreements even when the tasks did not require multiple approaches. S25 noted that their partner sometimes led them to incorrect solutions due to a lack of understanding. S6 explained that having a partner could be confusing, as conflicting ideas sometimes led to mixed results, whereas working alone allowed them to follow their own clear line of thought. These insights underscored the difficulties students encountered when collaborating with peers who employed divergent problem-solving strategies. Unresolved conflicts could impede progress on tasks, thereby hindering both learning and productivity. Persistent disagreements might also lead to frustration, creating a negative learning environment for both partners.

[S21]: *Sometimes, of course, it happens; there is a question, and we say this is the solution. He says something else and insists on it. For example, no one says it should be like this, no one says let's do this, and then if it doesn't work, we can try mine. I did this too. It happens sometimes.*

4.2.1.3.5 Seeking Assistance

Seeking assistance was identified as a crucial aspect of the learning process, particularly within collaborative instructional environments. Students reported frequently seeking help to overcome challenges, enhance their understanding, and improve their skills. Assistance could be sought from peers or teachers, with each source offering distinct benefits and addressing specific student needs. The category of seeking assistance encompassed the various methods by which students obtained support, the reasons behind their choices, and the strategies they employed for peer support.

Source of Assistance

Teachers or Peers as a Source of Support

Analysis results showed that in situations where group work proved ineffective or when students lacked group partners, they turned to alternative sources of assistance. Students indicated that they more frequently sought assistance from their teachers when they needed help. Students expressed that they sought assistance from their teachers ($f = 32$) more frequently than from their peers ($f = 14$) when they needed help. Eleven students expressed a preference for seeking guidance from their instructor, highlighting their trust in the teacher's expertise and commitment to providing support. Additionally, twelve students reported seeking help from classmates, reflecting their readiness to engage with the broader learning community. It is important to acknowledge that students demonstrated a range of approaches when seeking assistance. Some participants initially sought help from their peers, reflecting their preference for peer-to-peer learning. Others opted to seek guidance from the IT instructor directly.

[S14]: *I was asking for help from my friend, my group friend or my friend next to me in the line.*

Reasons for seeking assistance from peer

Teacher Unavailability

Teacher unavailability, highlighted by S5 and S27, emerged as a situation where students relied heavily on peers for support ($f = 3$). When immediate teacher assistance was limited due to factors like high student-to-teacher ratios or unexpected absences, students turned to classmates for help. These instances underscored the critical role of peer-to-peer learning as a complement to teacher instruction. Peers could provide immediate clarification, offer alternative explanations, and

collaborate on problem-solving, mitigating the impact of teacher unavailability and ensuring students could continue learning effectively.

[S5]: *When the teacher cannot help, we turn to those who can.*

Familiarity

The feedback provided by S7 and S19 emphasized the critical role of familiarity in facilitating effective peer-to-peer learning environments ($f = 2$). According to their statements, their preference for seeking assistance from friends stemmed from the shared routines, established connections, and sense of comfort that familiarity afforded. These connections promoted open communication and enhanced the learning experience.

[S19]: *The reason why I primarily seek help from my friend is that I feel more comfortable with myself because I am in the same place with my friend every day.*

Reasons for Seeking Assistance from Teacher

According to the results, students often sought assistance from teachers for various reasons that reflected the unique advantages teachers offered in the learning process. These reasons included the desire for clear explanations and guidance, the need for teacher expertise, and the support required for students with high self-perceived proficiency.

Clear Explanations and Guidance

Some students (S24, S28, and S29) indicated that they preferred the teacher's clear explanations and well-structured guidance ($f = 3$). The findings expressed by the students indicated that they found these explanations easier to follow and more comprehensive compared to those from peers who were at a similar learning stage. Student 29 explicitly mentioned that the teacher's ability to 'show it better directly'

and provide more explanatory instruction was particularly beneficial. This highlighted the teacher's expertise in crafting clear, organized explanations.

[S29]: *My teacher showed it better directly. ... My teacher explained it in a more detailed way.*

Teacher Expertise

S20 and S27 recognized the limitations of peer support for complex topics ($f = 2$). The findings indicated that students sought out teacher expertise when challenges required a deeper understanding of the subject matter than their peers could provide. This was particularly true for foundational concepts or intricate problems. This highlighted the vital role teachers play in student learning, as they possess a broader and deeper knowledge base that allows them to provide comprehensive explanations and guidance that peers may not be able to offer.

[S20]: *Because he is more knowledgeable about these issues.*

Self-Perceived Proficiency

The findings revealed that even students who perceived themselves as highly proficient, such as one participant who stated he was "ahead" of his peers, might still seek assistance from the teacher ($f = 1$). This situation underscored the crucial role teachers play in addressing the needs of all students, ensuring that even those who consider themselves advanced are appropriately challenged and supported despite their perception of mastering core programming concepts.

[S30]: *Since I usually go ahead of them, they are behind.*

Peer Support and Interaction

Participants indicated that they benefited from the diverse perspectives and knowledge of their peers, which enhanced their understanding and problem-solving skills. Nevertheless, findings showed that the effectiveness of peer support depended on the quality of interactions and the strategies employed. Constructive collaboration

fostered a deeper engagement with the material, while unproductive collaboration hindered learning progress.

Unproductive Collaboration Strategies

Student responses indicated that certain behaviors in peer support negatively impacted their learning, especially during plugged activities ($f = 16$). These behaviors mainly included showing their own solution for the peer to copy or complete the task for them (S10, S11, S14, S21, S24, S5, and S7). Student 21 mentioned the prevalence of simply copying a peer's solution to complete the task. While this seemed like a quick fix, it failed to promote genuine learning and skill development. Students 21 and 24 specifically mentioned that copying answers from their peers without understanding the underlying logic or problem-solving process did not significantly contribute to their learning. Student 21 added that they attempted to understand the solution afterward by reviewing it, but this highlights the limitations of this approach. Additionally, two students (S21, S26) expressed concerns about peers taking over problem-solving entirely instead of guiding them through the process to understand the concepts and develop their own solutions. This hindered the development of problem-solving skills and confidence in the struggling student. On the other hand, S21's comment, "I mean, of course, he postpones me a little bit, then he looks at my question." suggested that their peer's help was delayed, potentially hindering their learning progress.

[S21]: *I mean, of course, he postpones me a little bit, then he looks at my question. He tries the question he solved to do it himself first. If he cannot do it, he opens it from his own computer and gives it to me. ... I mean, when he does it there, of course, I can't understand it, I can't reason. But when he does it, I can say that I should have done it like this, for example, to find the answer to the question.*

[S14]: *If they passed that question, they would come back and show me that question. I couldn't understand it very well.*

Constructive Collaboration

Several students (S12, S13, S15, S19, and S26) mentioned receiving valuable hints and explanations from their peers when they encountered difficulties ($f = 9$). This support took various forms, such as clarifying concepts, identifying errors, suggesting improvements, and engaging in discussions. For example, S15 explained that their friend helped by explaining complex concepts. Some students indicated that their classmates assisted them by identifying errors in logic or code structure that they might have missed while working independently (e.g., S26 and S19). Others mentioned receiving hints on how to improve their code (e.g., S13). These helping approaches also demonstrated the peers' understanding of technical details and their ability to explain issues in a clear and actionable manner. Discussion was another helpful strategy, as highlighted by Student 12, who emphasized the value of learning through discussion and debate with peers. This approach encouraged critical thinking and challenged students to defend their approaches, leading to a deeper understanding of the concepts involved.

[S26]: *I would go to my friend's side with my teacher's permission and. For example, if you do this, you can do this; you did this wrong, you should have done it at this angle... Like that. I was helping with codes.*

[S12]: *I consult, I mean, by discussing with my friends because if I just listen to what they do, it would still be different, and I wouldn't understand.*

4.2.1.4 Theme 4: Independent Learning Approaches

While collaborative learning and seeking assistance were valuable strategies in programming education, the analysis of student responses also highlighted the importance of independent learning approaches. This theme focused on how students independently enhance their programming skills and understanding. These approaches included strategies such as utilizing guidance from coding learning environments, reviewing past solutions, engaging in trial and error, and self-

visualization. Furthermore, the results indicated that solo programming offers several benefits, including enhanced learning through active engagement, improved focus, and better retention of information. However, according to the students' statements, it also presented challenges, such as the lack of immediate peer consultation. Exploring these independent learning strategies provided insights into how students navigated their programming education autonomously. Table 4.17 displays the frequency with which participants identified codes related to the categories of the independent learning approach.

Table 4.17 Distribution of Code Frequencies by the Theme of Independent Learning Approaches

Categories	Codes	<i>f</i>
Independent Learning Strategies	Guidance from the coding platform	3
	Reviewing past solutions	1
	Trial and error	1
	Self-visualization	1
Benefits of Solo Programming	General positive perceptions	27
	Active engagement	8
	Improved focus	3
	Enhanced retention	2
Challenges of Solo Programming	Lack of pair consultation	18

4.2.1.4.1 Independent Learning Strategies

The results obtained from the interviews indicated four strategies employed by students while learning independently. These strategies were guidance from the coding platform, reviewing past solutions, trial and error, and self-visualization.

Guidance from the Coding Platform

The study demonstrated that students often relied on guidance from the coding learning environment as part of their independent learning strategies ($f = 3$). This guidance included hints and instructional videos that helped them understand and

solve coding problems. For instance, participant S27 expressed a desire for more hints, indicating their importance in the learning process. Similarly, S4 noted that videos appearing at the beginning and middle of coding tasks provided clearer explanations and made it easier to understand programming concepts. It was observed that these instructional resources were particularly useful when students encountered difficulties, allowing them to independently navigate challenges and enhance their problem-solving skills. The incorporation of these tools within the coding learning environment supported students' independent learning processes.

[S4]: *Yes, sometimes when we're just about to start coding, a video pops up at the beginning and then again in the middle, and we watch them. Watching them actually makes the explanations clearer. Without watching them, sometimes you can't understand what something is when it appears.*

Reviewing Past Solutions

The reviewing past solutions strategy identified instances where students employed an independent learning strategy centered on referencing past solutions ($f = 1$). S8 mentioned that he often revisited previously encountered code examples when faced with a programming challenge to find solutions and deepen his understanding of the subject matter. He added that he then searched for similarities between his current problem and the reference code, which involved either adapting a similar solution or skimming past irrelevant parts to explore a new approach. This strategy helped students by providing a foundation for developing new solutions to similar problems.

[S8]: *I go back to previous ones, I look at them. If there is something similar, I apply those.*

Trial and Error

Interview data showed that another approach that students engaged in was trial and error, as expressed by S24 ($f = 1$). This method captured instances where students mentioned experimenting with different problem-solving strategies and code variations until they found a solution. This highlighted a crucial aspect of

independent learning, emphasizing the importance of experimentation. By trying different approaches, students developed resilience in the face of challenges. They learned to troubleshoot, analyze outcomes, and refine their problem-solving skills. This highlighted a crucial aspect of independent learning, underscoring the importance of experimentation. By trying different approaches, students developed resilience in the face of challenges. They learned to troubleshoot, analyze outcomes, and refine their problem-solving skills.

Self-Visualization

The fourth approach, self-visualization, as described by Student 16, involved using mental visualization to understand problem-solving processes or potential solutions ($f = 1$). This strategy enhanced problem-solving skills by encouraging students to think through different approaches and plan problem-solving steps before coding. These independent learning strategies enabled students to assume responsibility for their learning and develop valuable skills.

[S16]: *In situations like these, I would imagine myself. For example, those things, squares, you know, I would feel like I was in the squares and determine which way to turn. I would imagine myself in the same place and decide where to go. And it became very easy.*

4.2.1.4.2 Benefits of Solo Programming

Students' statements showed that solo programming, the practice of working independently on coding tasks, offered several advantages that could enhance the learning experience for students. Based on student responses, the key benefits highlighted by student feedback were improved focus, enhanced learning through active engagement, and enhanced retention.

General Positive Perceptions

Nearly half of the students expressed a preference for solo programming in their statements ($f = 27$). This indicated that a significant portion of the student participants recognized the autonomy and self-reliance that solo programming offers. Additionally, six students (S3, S12, S24, S25, S28, and S30) mentioned the absence of disadvantages in solo programming and its potential to positively impact their learning.

Active Engagement

This code captures feedback where students expressed that they learned more effectively when working alone because they were more conscious, responsible for all aspects of problem-solving, and focused on understanding the material deeply ($f = 8$). Eight of the interviewed students (S7, S8, S13, S16, S17, S22, S23, and S24) emphasized the value of solo programming as a tool for fostering deeper learning and understanding. Their statements showed that solo programming encouraged active engagement by placing the onus of learning squarely on the individual (S8). This shift in responsibility led to several positive outcomes. S13 reported feeling more conscious of his learning process when working alone and being more aware of his own strengths and weaknesses, allowing him to focus on areas that required improvement. Analysis results showed that solo programming encouraged students to take ownership of the problem-solving process. They were forced to analyze concepts, identify solutions, and implement their ideas independently. This active engagement led to a more profound understanding of the underlying principles, as noted in S16's statement. Working alone also allowed students to identify and rectify their mistakes without the immediate intervention of others. This process of self-correction reinforces learning and promotes a growth mindset, as demonstrated by S24's experience. In conclusion, solo programming emerged as a powerful tool for fostering active engagement and enhancing learning outcomes in programming education.

[S_8]: *I couldn't understand the codes he mentioned because I couldn't grasp what the code was and how it worked without looking at it myself. But when I looked at it myself, I understood better.*

[S16]: *The positive side of coding alone is that you can see all the questions and answer them yourself. You try to solve them, engage your brain a bit, and I think it's better.*

Improved Focus

Students (S17 and S26) mentioned that they were able to concentrate better and listen to the teacher more attentively when they were working alone, without the presence of peers causing distractions ($f = 3$). Besides that, students added that they struggled with tasks because they were not paying attention during the lesson due to talking with their pairs. This finding highlighted the positive impact of solo programming on focus and attentiveness.

[S17]: *I would be more open in the informatics (Information Technologies and Software) class. How can I put it? I would sit calmly and listen to the lecturer. There would be no one next to me. I get distracted.*

Enhanced Retention

Solo programming appeared to contribute to improved information retention and long-term learning, as highlighted by S13 and S14 ($f = 2$). According to students' statements solo work led students to attempt to solve more problems independently. The findings revealed that this increased practice and exposure to the material could further solidify students' understanding and enhance their ability to recall information later.

[S14]: *...but it was more memorable. Because you were solving more questions.*

4.2.1.4.3 Challenges of Solo Programming

Lack of Pair Consultation

A significant portion of the students highlighted that the lack of peer consultation was a major drawback of working alone on programming tasks ($f = 18$). They emphasized the importance of immediate support and collaboration, particularly when encountering complex problems. S26 expressed frustration with the inability to get assistance when unable to complete tasks on their own. Similarly, S15 mentioned leaving tasks unfinished due to the inability to find solutions on their own. According to participant S19, the absence of peer support could hinder progress, particularly when everyone was focused on their individual tasks. This limitation was further amplified when instructors were unavailable for assistance. S21 also pointed out that the absence of different perspectives limited their problem-solving approach. Additionally, students S9, S29, and S30 emphasized the need for a peer to provide fresh ideas and guidance when progress stalls while working alone. These insights underscored the importance of peer consultation in the learning process. The ability to collaborate and seek immediate feedback from peers could significantly enhance problem-solving capabilities and overall learning outcomes. Analysis results showed that without this support, students struggled to overcome challenges, leading to frustration and incomplete tasks.

[S22]: *...for example, when I ask the teacher about a subject I don't understand, sometimes I can't understand it, I can't find out what to do. When I had a friend, he helped me, we could find it together, but when he wasn't, I had some difficulty.*

4.2.1.5 Theme 5: Goal Setting

In this study, student responses emphasized goal setting as a crucial element of the learning process, significantly influencing their approach to and engagement with learning. The findings revealed that more than half of the students expressed a desire

to master programming skills. This theme explored the different types of goals participants set for themselves and how these goals impacted their motivation and learning strategies in programming education. As seen in Table 4.18, the codes identified through qualitative data analysis underscored the varied motivational orientations of students, encompassing mastery-oriented goals, performance-oriented goals, and performance-avoidance goals.

Table 4.18 Distribution of Code Frequencies by the Theme of Goal Setting

Categories	Codes	<i>f</i>
Mastery-Oriented Goals	Career-oriented goals	44
	Challenge seeking	18
	Desire to simplify complex tasks	12
	Daily life context relevance	12
	Recreational interest in coding	8
Performance-Oriented Goals	Completion-driven motivation	11
	Competition focus	4
	Academic achievement focus	1
Performance-Avoidance Goals	Avoidance of challenging tasks	20
	Fear of failure	6
	Skipping tasks	5

4.2.1.5.1 Mastery-Oriented Goals

This category explored the intrinsic motivation participating students had to learn, understand new concepts, and master the skills in programming education. The codes under this category reflect the various ways students approach their learning in programming education with a mastery-oriented mindset, emphasizing deep understanding and long-term skill development.

Career Oriented Goals

Half of the participants articulated the significance of programming for their personal development and future careers, recognizing its essential role in today's technology-driven world ($f=44$). This career-oriented motivation served as a significant impetus for their learning, as students understood programming as an essential tool for

achieving their professional goals. They identified the value of programming skills in various fields, including software engineering, computer engineering, IT, and game development (S12, S13, S18, and S20). Furthermore, ten students highlighted the importance of coding skills, acknowledging their potential to unlock various career paths. They expressed a strong belief that coding could be a critical skill, enhancing their employability and adaptability in the professional world. This positive perception could serve as a strong motivator for students to continue learning and pursuing their programming goals.

[S20]: *Because the future profession I think about is software engineering. That's why I pay attention to it. That's why I'm interested in software... I try to choose software because I'm interested in it.*

[S1]: *I mean, I think it will be important when I grow up, when I have a profession. I already think it will be... I mean, when I grow up now when I get into jobs, coding will be in jobs because it happens a lot. I'm not sure right now, so it will definitely be coding when I grow up.*

Challenge Seeking

This code captured student feedback that demonstrates a desire for intellectual growth and a preference for learning experiences ($f = 12$). Students expressed their preferences for stimulating and demanding tasks, highlighting their mastery-oriented goals. This preference showed that students were intrinsically motivated to learn, seek challenges to improve their skills, and strive for mastery over a subject. While nine of the students expressed their enjoyment of overcoming difficulties, S12 and S30 conveyed dissatisfaction with tasks in the digital coding learning environment they perceived as too easy, indicating a positive attitude towards challenges and a preference for intellectually stimulating material. For instance, S1 enjoyed the challenge of placing colors together, and S15 found the flowchart difficult yet fun. S21 emphasized the satisfaction gained from completing challenging tasks, like puzzles and brain teasers. Similarly, S7 stressed the excitement that comes with increasing difficulty. This is further reinforced by

students like S12, S9, and S30, who expressed a preference for more challenging tasks and a dislike for overly simple ones. Overall, the students' comments reflected their desire for intellectual challenges and their enjoyment of overcoming difficulties, indicating a strong focus on mastery and growth in their coding journey.

[S7]: *When things get progressively harder, you get even more excited.*

Desire to Simplify Complex Tasks

Twelve statements from the students suggested a strong desire to deeply understand the concepts behind the tasks rather than merely completing them. This reflected a mastery-oriented approach to learning. Students actively identified challenging parts and sought ways to manage their cognitive load. For instance, S10 expressed a preference for text explanations in flowcharts over visual flowcharts, citing difficulty with understanding flowchart symbols (as discussed under the category of 'inherent complexity of concepts and tasks'). This preference indicated that text descriptions, particularly writing out the algorithm as a series of steps, provided the clarity needed to comprehend the underlying algorithm, making the concept more manageable. Similarly, the other two students (S2 and S18) expressed a desire to simplify difficult topics altogether. This inclination towards simplification suggested that students were employing specific learning strategies, aiming to break down complex tasks into manageable components, ultimately leading to a deeper understanding of core coding concepts.

[S10]: *Teacher, I would like to change the things in the flowcharts, the visuals. I would prefer them to be written in text, not with shapes.*

[S18]: *I would like to change the nested loops, teacher. I am very bad at that. I would like to remove that topic.*

Daily Life Context Relevance

Based on the analysis, even students with less defined career goals acknowledged the long-term value of programming in various aspects of daily life ($f = 12$). Eight students (S3, S10, S15, S18, S21, S26, S27, and S30) emphasized the applicability

of programming beyond professional settings, recognizing its potential to simplify tasks, solve problems, and enhance general understanding in daily life. For example, S15 illustrated how coding concepts like decision-making algorithms and conditional statements could be applied to real-life scenarios to help structure decision-making processes and optimize choices in everyday situations. Student 27 highlighted the potential for programming to enhance their life by creating algorithms or writing down instructions, effectively automating or simplifying tasks like cooking or other household chores. Additionally, students recognized the potential for programming to contribute to future technological advancements. The responses of three students (S10, S18 and S26) suggested that they saw coding as a valuable tool for understanding the technological world around them. This included comprehending how technology is used in everyday devices, apps, and services and developing an informed perspective on the impact of technology on society. Overall, the student feedback indicated that programming is not just a technical skill but also has the potential to enhance various aspects of daily life. According to results of the analysis, this broader understanding of programming's relevance could serve as a motivator for students to continue learning and explore its applications in their personal and social spheres.

[S27]: *It is important to me. It can help me in difficult situations in my life. For example, if my mother is going to cook and says, 'Do it yourself, I'm leaving,' I can ask her to create an algorithm for me. She would ask, 'What's an algorithm?' Then I would explain it to her, and she would do it for me. Then I can do it myself.*

[S10]: *Teacher, it can be useful in technological devices. For example, in America, we can call Teslas by phone. In that respect, I think it is necessary.*

Recreational Interest in Coding

Not all students approach coding with a purely career-oriented mindset. This section explores the motivations of students who viewed coding as a fun and engaging activity, separate from professional aspirations (S2, S3, S11, and S16). These

students find enjoyment, entertainment, and creative potential in the learning process itself. For some, like S11, the inherent satisfaction and enjoyment derived from learning code is the primary motivator. Others, like S2 and S16, highlight the entertainment value of coding, suggesting it provides a pleasurable learning experience. Furthermore, S3 views coding as a potential hobby, offering a creative outlet for their free time. These responses highlight how coding can be perceived as a source of personal enjoyment and creative exploration. For these students, the intrinsic pleasure of coding, rather than its potential career benefits, is the primary driver of their engagement. This recreational interest underscores the importance of fostering a learning environment that recognizes and supports diverse motivations for learning coding, ensuring that it remains accessible and enjoyable for all students, regardless of their professional aspirations.

[S3]: *I only do it as a hobby. I will do it as a hobby in the future.*

4.2.1.5.2 Performance-Oriented Goals

According to the findings, the performance-oriented goals of the students were driven by external factors such as completion-driven motivation, competition, and the desire for academic achievement. Students with these goals were often motivated by the need to outperform others and gain recognition. This external motivation often led to a focus on achieving high grades, excelling in tasks, and receiving praise or recognition from teachers and classmates.

Completion-Driven Motivation

Responses from students (S4, S5, S7, S14, and S21) indicated a strong motivation to achieve specific performance goals, such as completing all levels or tasks within a gamified learning environment ($f = 11$). The digital coding learning environment used in their coding education lessons featured puzzle sets and levels that turned green upon completion. As students advanced through levels, each completed level was marked by a green circle, providing a visual representation of progress.

Achieving these targets elicited a sense of accomplishment and satisfaction, highlighting the students' intrinsic motivation to meet challenges and attain a sense of completion. This finding underscored the positive influence of gamification and goal setting on enhancing student engagement and motivation.

[S4]: *Sometimes we do something. We log into my friend's code.org account who sits next to me, and then we log into my account. When we use his account, I complete the parts I haven't done at home.*

Competition Focus

A competitive focus emerged within the learning environment, as evidenced by the statements of students S1 and S6 ($f = 4$). Their primary objective was to complete tasks quickly and potentially surpass others, prioritizing speed over balanced participation and collaborative learning. This performance-oriented approach highlighted a potential pitfall in student motivation, where an emphasis on external validation through competition can overshadow the intrinsic value of learning. For instance, S1 readily conceded control of the task, seemingly motivated by surpassing others. Similarly, S6 focused on personal advancement, framing their actions within the context of progressing their individual account. This emphasis on individual achievement could hinder the development of a growth mindset and a deeper understanding of the material.

[S1]: *...Then he said, 'Let me do it so that we can beat the others so that we can do it faster. I said okay.*

Academic Achievement Focus

One of the students' perspectives shed light on the positive influence of aligning coding tools with assessment practices ($f = 1$). S24 highlighted how the teacher's use of coding-based tasks and questions in exams directly mirrored the activities conducted within the coding learning environment. This close connection served as a motivator for performance-oriented students like S24, who prioritize academic achievement, as it provided a clear path to attaining high scores. This finding

underscored the importance of meaningful assessment practices that were directly connected to students' learning activities. When coding exercises and tools were demonstrably relevant to exams, they encouraged students to actively engage with the material and strive for mastery. This fostered a performance-oriented focus that was channeled toward developing strong coding skills.

[S24]: *For example, when the teacher gives an exam, he always asks coding questions. He asks questions through coding. He gives such shapes on the exam paper. For example, we do the same activities and the instructor asks the same questions, like that. Therefore, it provides me with a benefit in that respect It also helps me get high scores on exams.*

4.2.1.5.3 Performance-Avoidance Goals

This category explored student feedback that highlighted a tendency toward performance-avoidance in programming education. The theme centered on students' focus on avoiding negative performance outcomes, negative judgment, and comparison rather than striving for mastery or intrinsic learning. Students driven by these performance-avoidance goals often prioritized strategies to minimize the risk of failure rather than actively seeking challenges to enhance their learning.

Avoidance of Challenging Tasks

Data analysis results showed that participants often expressed a preference for avoiding challenging tasks, highlighting their reluctance to engage with difficult programming concepts and activities ($f = 20$). This aversion was explicitly stated by six students (S6, S8, S23, S24, S27, and S28) who preferred easier tasks, while another six (S2, S9, S14, S18, S20, and S22) expressed a dislike for challenging tasks. These challenging tasks included concepts like conditional statements, nested loops, and variables. These findings suggested that task avoidance, potentially driven by a desire to minimize negative emotions associated with difficulty, was a significant issue in programming education.

[S24]: *I can say that my least favorite thing is difficult coding.*

Fear of Failure

Several participants (S2, S7, S21, and S28) disclosed a fear of making mistakes or failing, which led them to avoid participation in programming tasks ($f = 6$). This reluctance stemmed from anxieties about negative judgment, embarrassment, or failure in front of others. For example, S7 described feeling ashamed of not being able to perform adequately in a large group setting. Similarly, S21 expressed frustration when his code came out wrong, showcasing the discouragement that mistakes can bring. These student experiences served to underscore the potential obstacles posed by the apprehension of failure.

[S7]: *Because there were many people around. I was ashamed when I couldn't do it.*

[S28]: *I can't think of the name, but some things were difficult, teacher. I was afraid that I couldn't do it.*

Skiping Tasks

Analysis results showed that two students (S14 and S21) revealed a tendency to skip tasks or problems they perceived as too difficult, employing this as a performance-avoidance strategy ($f = 5$). This highlighted a potential performance avoidance approach to learning, where students prioritized avoiding negative emotions over actively engaging with challenging material. By skipping these tasks, they might have missed out on crucial learning opportunities and potentially hindered their overall progress.

[S21]: *Mostly, if there were three of us or two of us, we would say, let's skip it. For example, there were many examples we couldn't do. I think we skipped all of them.*

4.2.1.6 Theme 6: Affective Aspects

The purpose of this theme was to examine the emotional and attitudinal aspects of learning programming. This theme extended beyond the acquisition of technical skills and knowledge. It explored the feelings, beliefs, and motivations that influence students' engagement and success in programming education. The affective factors were examined through two primary aspects: attitude and self-efficacy. Related categories, sub-categories, and their codes, along with the frequency of participant responses, are detailed in Table 4.19.

4.2.1.6.1 Attitude

This category focused on students' dispositions towards programming, encompassing both positive and negative attitudes that shape their learning experiences. Positive attitudes included an interest in learning programming, enjoyment of both plugged and unplugged activities, the appeal of familiar characters, the satisfaction derived from social interactions, a favorable view towards gamified learning, and a positive classroom atmosphere. Conversely, negative attitudes involved a general negative disposition towards programming and frustration from prolonged use and.

Table 4.19 Distribution of Code Frequencies by the Theme of Affective Aspects

Categories/Sub-Categories	Codes	<i>f</i>
Attitude		
Positive Attitudes	Interest in learning programming	54
	Enjoyment of plugged activities	30
	Appeal of familiar characters	17
	Enjoyment of social interaction	14
	Engagement of gamification	9
	Enjoyment of unplugged activities	8
	Positive classroom atmosphere	6
Negative Attitudes	Frustration from prolonged use	14
	Negative disposition towards programming	2
Self-Efficacy		
Confidence in Coding Abilities	Low	11
	Moderate	18
	High	27
Determinants of Self-Efficacy Perceptions	Mastery experiences	12
	Social recognition from peers	4
	Peer comparison	4
	Perceived cognitive abilities	4
	Academic performance	2

Positive Attitudes

Interest in Learning Programming

The findings revealed a predominantly positive sentiment towards programming, with a significant majority of students (26 participants) expressing interest and enthusiasm for the subject ($f = 54$). Many students highlighted their fascination with coding, emphasizing its intriguing and intellectually stimulating nature. For instance, students S10 and S12 demonstrated a high level of intrinsic motivation and curiosity about programming. Students consistently described their programming lessons as engaging and interesting, and Student 18's mention of enthusiasm underscored the widespread appeal of programming. Overall, the positive feedback from students illustrated a pervasive interest in learning programming, driven by its engaging and

intellectually stimulating aspects, which highlights the effectiveness of programming education in fostering lasting enthusiasm for the subject.

[S18]: *It's really captivating, teacher. I love coding.*

Enjoyment of Plugged Activities

The student feedback overwhelmingly highlighted a strong positive association with plugged activities, suggesting a high level of intrinsic motivation and engagement ($f = 30$). Seventeen out of thirty students explicitly mentioned enjoyment, fun, and high levels of engagement with the activities on the digital coding platform (Code.org). This underscored the importance of the emotional and affective aspects of the learning experience. One student expressed a desire to conduct all classes in the computer lab, suggesting a preference for technology-integrated programming education. Additionally, S3 and S7 mentioned enjoying being taken to the computer lab, further indicating a positive attitude towards tech-enhanced learning environments. These student voices emphasized the positive emotional response elicited by plugged activities. Analysis results showed that when learning was perceived as enjoyable and engaging, students were more likely to be intrinsically motivated and maintain their interest throughout the learning process.

[S3]: *Going down to the computer lab was really better for me as well. We have fun. We receive education in the computer classroom.*

[S27]: *So, when you enter, you feel like doing it. When you look at the questions, you feel like doing them. Because there were nice questions. There was good coding and all.*

Appeal of Familiar Characters

The use of sprites within the digital programming learning environment elicited mixed responses from students ($f = 17$). Five students (S6, S8, S9, S14, and S27) expressed positive attitudes towards the learning environment that integrated characters they recognized and enjoyed from other media. The findings suggested that the integration of well-known figures into the digital programming environment

contributed to a more positive learning experience by enhancing students' interest and enjoyment. However, the study also revealed a need for continuous improvement and diversification of these characters. Three participants (S1, S8, and S22) expressed a desire for more engaging and relatable characters, suggesting that the existing options may not resonate with all learners. This underscored the importance of incorporating a wider range of characters from different sources, along with the ability to personalize characters, which could improve student engagement and overall satisfaction with the coding program.

[S8]: *I mean, I was more interested in it because it had such well-known game characters and so on, so I did it more easily. It made it easier, having characters both excited and made it easier.*

[S22]: *About coding, you know Angry Birds, there could have been other films. For example, Bumblebee or something like that would be better about robots. It would be more fun, so there would be a difference. It would be more fun.*

Enjoyment of Social Interaction

This code examined the social interaction aspects of programming education ($f = 14$). Unplugged activities, which involved hands-on, non-digital tasks, emerged as a contributor to social interaction in programming education. S14 specifically mentioned enjoying socializing and working with friends during unplugged activities and emphasized a preference for unplugged learning activities that offer more opportunities for social interaction and collaboration. Pair programming, on the other hand, was typically associated with computer-based programming activities. While pair programming also fostered social interaction and collaboration, it differed from unplugged activities in that it was specifically focused on programming tasks and utilized digital tools. Students (S4, S10, S18, S24, and S26) expressed their enjoyment of working together with friends, indicating a preference for cooperative learning environments. Students mentioned that collaborative learning provided opportunities to foster improved communication and teamwork skills, as noted by

S24 and S26. Additionally, seven students (S2, S6, S10, S14, S18, S24, and S26) expressed a preference for collaborative programming over solo programming, finding solo programming less enjoyable and more tedious due to the lack of social interaction and conversation that accompanied working independently.

[S14]: *But the things we did by socializing were also good. For example, in some classes, we went out to the schoolyard and... At one point, the IT teacher brought something to our classes, a rabbit hole thing, a rabbit hole. The rabbit was trying to reach the carrot. For instance, drawing a larger version of that on the ground in the schoolyard and playing with it.*

[S26]: *I socialized a little more there. He also liked coding like me. I mean, if it wasn't for the computer, we wouldn't have met him.*

Engagement of Gamification

The qualitative analysis revealed a positive student response (S1, S2, S3, S4, S14, and S17) toward the integration of gamified elements within the programming learning environment ($f = 9$). This positive reception underscored the potential of gamification to enhance both student engagement and motivation. Students like S1 and S3 specifically highlighted the enjoyable nature of Code.org games, suggesting that engaging gameplay mechanics effectively capture student interest. This aligned with the concept of "flow" in gamified learning, where students were intrinsically motivated and fully absorbed in the learning process. Additionally, the overall game-like approach, as described by student S17, contributed to a more enjoyable learning experience, thereby reducing the perceived difficulty associated with coding. S4 also noted the effectiveness of gamified elements in promoting active cognitive engagement. These student experiences provided evidence for the efficacy of gamified learning in fostering student engagement and motivation within programming education.

[S3]: *Anyway, the games we played on code.org were very fun. I mean if we think about it, we actually write coding, but it was really fun. That 'go forward' or finishing the game. These were really fun in coding.*

Enjoyment of Unplugged Activities

While not as numerous as for plugged activities, some students (S4, S5, S7, S22, S26, and S27) also reported enjoyment and high levels of engagement with unplugged activities ($f = 8$). These activities often involved hands-on, collaborative tasks that did not require digital devices. Student statements reflected feelings of satisfaction, indicating that the unplugged activities were engaging and enjoyable. These positive responses highlighted the affective benefits of unplugged activities and their contribution to a positive learning environment.

[S4]: *I really liked that glass game. We also did something like this, we moved like a robot. The teacher wrote it on the board. We had turned our backs. One of our friends came out. One of them was a robot and the other one was saying what was on the board. There were degrees, he said to stay there, he said to turn right-left. It was a lot of fun. It was good.*

Positive Classroom Atmosphere

The study also revealed the importance of positive teacher behavior in fostering a positive classroom atmosphere ($f = 6$). Several students (S1, S5, S6, and S12) specifically commended their teacher's politeness, kindness, and calm demeanor. S1 indicated the teacher's gentle explanations and noted the absence of yelling, even in frustrating situations. This positive and respectful approach was highly valued by S1 and S6 and was seen as crucial to fostering a more engaging learning environment by S12. Ultimately, it was seen that the teacher's positive behavior cultivated a sense of trust and mutual respect, establishing a classroom culture that promoted both academic learning and personal growth.

[S1]: *Also, the teacher is kind. He explains things kindly. I am happy because he doesn't yell. It's the first time I've seen this. He only yells if we make him very angry, and even then, it passes quickly. I really like our teacher. When we say something, he says okay.*

Negative Attitudes

Frustration from Prolonged Use

Several students (S4, S6, S9, S15, S21, and S23) reported experiencing fatigue or frustration as a result of prolonged use of specific platforms or instructional methods ($f = 14$). Students such as S9, S15, and S21 expressed a desire for increased variety in instructional approaches and activities to maintain their engagement. For instance, participant S4 found that moving from one Code.org course to another without sufficient variation became monotonous, highlighting a need for more diverse and stimulating activities. This highlighted the importance of integrating a diverse range of activities and learning environments into the curriculum to prevent student burnout and sustain engagement.

[S4]: *For example, when you finish one course and move on to the next, it gets a bit overwhelming. It really becomes boring.*

[S9]: *I would like to try other new things.*

Negative Disposition Towards Programming

Two students (S1 and S19) expressed an overall negative outlook toward learning programming ($f = 2$). Their feedback highlighted significant challenges and a general lack of enthusiasm for the subject, in contrast to the more positive responses from other students. S1 succinctly conveyed his negative disposition towards programming. Similarly, S19 articulated difficulties with programming, reflecting their struggle and lack of engagement with programming tasks.

[S19]: *And also, I didn't like coding. I didn't quite understand it. It was a bit difficult.*

4.2.1.6.2 Self-Efficacy

In this study, within the context of programming education, this category encompassed two primary aspects. The first aspect was confidence in coding abilities, which highlights the level of self-efficacy students perceived in their coding skills. The second aspect was the determinants of self-efficacy perceptions, which investigated the various influences on students' self-efficacy.

Confidence in Coding Abilities

Through in-depth interviews, students revealed a range of self-efficacy in coding, with some exhibiting high confidence and others struggling. Categorizing their responses, it was found that five students exhibited low self-efficacy ($f = 11$), twelve demonstrated moderate self-efficacy ($f = 18$), and thirteen displayed high self-efficacy in their coding abilities ($f = 27$). Students who expressed high self-efficacy often described coding as easy or simple (e.g., S29, S13, and S25). Their comments highlighted the confidence and comfort that high-self-efficacy students associated with coding. While some students expressed overall ease in coding, others acknowledged that some tasks or concepts were challenging. Students with lower self-efficacy, like S1, reported difficulty and frustration, highlighting the varying levels of self-efficacy and perceived difficulty among students.

[S29]: *Actually, I didn't have any difficulty. It was all very easy.*

[S26]: *I also realized that this job is hard. It's not an easy job.*

Determinants of Self-Efficacy Perceptions

This sub-category examined students' reflections on their self-efficacy and explored the factors that influenced their perceptions of self-efficacy in programming. Several key determinants emerged, including mastery experiences, social recognition from peers, peer comparison, perceived cognitive abilities, and academic performance.

Mastery Experiences

Findings showed that successfully completing coding exercises or challenges, regardless of difficulty, fostered a sense of accomplishment in students (S2, S5, S8, S9, S12, S20, S25, S27, and S29), ($f = 12$). This positive reinforcement, exemplified by S8 feeling like a "programmer" after completing a task, built confidence and contributed to a strong sense of self-efficacy in problem-solving. This finding exemplified the connection between successful problem-solving and confidence. When students experienced the satisfaction of completing tasks, they developed a sense of competence and a belief in their ability to achieve future challenges. Additionally, the findings suggested that the duration of task completion played a role in students' perception of their programming success (S9 and S26). Students who completed tasks quickly tended to feel even more successful in coding. However, struggling with tasks could lead to frustration and potentially hinder self-efficacy, as observed in students like S21 and S24.

[S9]: *Because once, while the teacher was explaining, I understood the topic. I completed it in no time. That's when I realized I was successful, considering how quickly I did it.*

Social Recognition from Peers

One key factor affecting self-efficacy perceptions in programming education was identified as social recognition from peers ($f = 4$). The study found that students who received help requests from their classmates regarding coding tasks demonstrated greater confidence in their abilities (S8 and S26). This positive reinforcement from

peers contributed to a strong sense of self-efficacy. For example, student S26 described feeling like a professor when helping classmates, which significantly boosted his self-confidence. Similarly, student S8 shared that his classmates frequently sought his help, which also enhanced his self-efficacy. These findings indicated that peer interactions and the opportunity to assist others reinforced a student's belief in their coding skills.

[S26]: *For example, when we log in, I somehow feel like a professor. I feel like someone who has become an expert in these things. My friends ask me questions, and I tell them, 'You can do it this way.' At those times, I feel really good.*

Peer Comparison

Analysis results showed that peer comparison played a significant role in shaping self-efficacy or belief in participants' ability when learning to program ($f = 4$). As a result of the social nature of learning, students compared their skills and performance to their peers, impacting their self-efficacy. The data showed that three students were influenced by this phenomenon. For instance, participant S26 felt a sense of accomplishment by observing their classmates struggle with a particular section, contrasting it with their own progress. On the contrary, S7, despite acknowledging his achievements, felt inadequate compared to his stronger peers. This comparison-based assessment reinforced their belief in their abilities and contributed to their overall self-efficacy in programming.

[S7]: *I see myself as successful, but I can't say I'm very good. Because there are others who are better than me. I'm not at their level. Just a bit above average.*

Perceived Cognitive Abilities

Statements from students revealed a connection between their self-assessment of cognitive skills (thinking and learning abilities) and perceived self-efficacy ($f = 4$). S1, S8, S17, and S29 discussed how their thinking and learning abilities influenced

their confidence in coding. Those who believed they had strong thinking and learning skills, like S29, tended to feel more confident in their ability to learn and succeed in coding. Conversely, those who doubted their cognitive abilities, like S1, were more likely to experience lower self-efficacy, potentially leading to struggles with motivation and engagement in coding. This highlighted the importance of self-perception in students' motivation and engagement.

[S29]: *Because my understanding capacity is higher...*

[S1]: *Because my brain couldn't take it in much...*

Academic Performance

Students' comments also illuminated the relationship between course grades and self-efficacy ($f = 2$). Students like S25, who mentioned good grades in their IT and Software courses, perceived them as validating their coding abilities. This highlighted the potential of academic performance to act as positive reinforcement. The findings indicated that strong grades could enhance self-efficacy, motivating students such as S6 to persist in their learning and embrace new challenges in coding.

[S6]: *I received a score of 100 on three assignments. I know from that I'm good at coding*

CHAPTER 5

DISCUSSION AND CONCLUSION

The aim of this study was to investigate the factors influencing middle school students' learning of programming fundamentals. To address this aim, the primary research question, '*What factors influence the acquisition of fundamental computer programming concepts in fifth-grade students?*' was examined. To further contextualize the inquiry, five sub-research questions and their corresponding sub-questions were also examined. In this mixed-methods study, the quantitative and qualitative data were analyzed independently. The major findings from both data sets were then discussed within the framework of the research questions, considering the variables under investigation. Following this discussion, the chapter concluded with a synthesis of the key findings. Finally, the implications of the results and directions for future research were outlined.

5.1 Major Findings and Discussion

5.1.1 Cognitive Load

The study revealed that extraneous load significantly predicted coding performance. Although germane load was not found to be a significant predictor, its substantial correlation with extraneous load necessitated its exclusion from the regression model. As a result, the independent contribution of intrinsic load to coding achievement could not be assessed.

The study findings indicated that students encountered their most substantial cognitive load, both intrinsic and extraneous when engaged in learning the nested-loop concept. Loops were also identified as the third most challenging concept in terms of intrinsic cognitive load and the fourth most challenging in terms of

extraneous cognitive load. The research findings indicate that students' intrinsic cognitive load was significantly higher during the week when they learned about loops and nested loops compared to the weeks focusing on conditionals, variables, and testing and debugging. Furthermore, it was found that the cognitive load associated with nested loops was significantly higher than that associated with conditionals, variables, and testing and debugging. While students experienced a relatively high level of germane load during the week dedicated to nested loops, the overall increasing trend of germane load across weeks exhibited a decline, specifically for the nested loops topic. However, this decline in germane load for nested loops was not statistically significant.

Interview data also revealed that participants perceived nested loops as a more significant challenge than simple loops. The concept of nested loops was the third most commonly highlighted topic within the thematic category of "inherent complexity of concepts and tasks". These results align with the existing literature, which categorizes the learning of simple loops and nested loops as some of the most challenging foundational programming concepts for novices, both at higher education levels (Gomes et al., 2019; Winslow, 1996) and in elementary education (Grover & Basu, 2017). In the present study, participants reported difficulties in determining the number of iterations for each code block within nested loops, particularly when the total number of loops increased. Consistent with this study's findings, Gomes et al. (2019) reported that students in their CS1 course encountered greater difficulties with internal loops compared to external loops, particularly when the external loop completed its second iteration. Similarly, participants in this study reported difficulties in determining which code blocks would be executed when curly braces ({}) were omitted. Although in the current study, code blocks for both inner and outer iterations were visually represented, novice programmers may still need help comprehending the hierarchical structure of nested loops and the order of execution. This suggests that understanding the hierarchical structure of nested loops and the order of execution is a common challenge for novice programmers.

Sleeman et al. (1984) further clarified the challenges students face in comprehending loops by emphasizing the difficulties in understanding the role of the control variable within loops. This study emphasizes the cognitive challenges associated with comprehending the iterative nature of loops and the management of loop variables. While constructs such as loops are often associated with procedural programming, their significance in object-oriented languages like Java underscores the hybrid nature of modern software development. The challenges encountered by programmers in mastering these constructs highlight the need for a comprehensive approach that encompasses both procedural and object-oriented concepts (Dale, 2006). The findings of another study suggested that while sixth-grade students encountered some difficulties with loop concepts, the visual nature of the Scratch environment may have reduced some of the cognitive challenges typically associated with programming. Additionally, the positive impact of prior experience with digital tools on students' ability to adapt to the Scratch interface highlights the importance of providing students with opportunities to engage with technology from an early age (Çakiroğlu et al., 2018).

Building upon the challenges presented by nested loops, basic sequences emerged as the second most demanding concept in terms of cognitive load. The intrinsic cognitive load associated with basic sequences was significantly greater than that associated with conditionals, variables, and testing and debugging. However, significant differences in extraneous cognitive load were only observed for testing and debugging when learning basic sequences. Similarly, the germane load experienced for this topic was significantly lower than for variables and conditionals. These results could be attributed to students' perceptions of coding during the initial week. The singular occurrence of the code pertaining to "sequencing and logical flow difficulties" in the qualitative data provides additional evidence to support this interpretation. This finding can be explained by the unfavorable attitudes held by the students toward coding, stemming from their limited exposure to programming before commencing the course (Çakiroğlu et al., 2018). On the other hand, as proficiency and familiarity with the learning environment increase, a learner can

reduce their cognitive load (Sweller, 2010). The empirical evidence from this study, characterized by an upward trend in germane load across most topics, excluding nested loops, aligns with the proposed explanation. Inexperienced learners are more likely to give up on learning computer programming if the task they are attempting to complete is complex. Inexperienced learners are more susceptible to experiencing cognitive overload and frustration in the context of learning programming (Bounajim et al., 2021). The high extraneous cognitive load observed in the first week suggests that the unplugged activity may not have been entirely clear to the students. The complexity of the task may result in cognitive overload, which may interfere with the performance of the task and/or the learning of the subject matter. This interpretation is further supported by the students' statements during interviews, which indicated difficulties in understanding and implementing the unplugged activity due to unclear task instructions. Besides that, findings of the related literature indicate that block-based coding platforms such as Scratch and code.org make coding easier, particularly by preventing children from encountering syntax errors (Resnick et al., 2009). The findings that such coding environments reduce extraneous cognitive load can be associated with the lower cognitive load of students in plugged activities in this study (Meerbaum-Salant et al., 2013). This study's observation of higher cognitive load during unplugged activities may be partially explained by findings from previous research. Studies have shown that block-based coding platforms like Scratch and code.org simplify coding, particularly by preventing children from encountering syntax errors (Resnick et al., 2009). By eliminating the need to focus on syntax, these platforms are thought to reduce extraneous cognitive load (Meerbaum-Salant et al., 2013). Considering the additive nature of types of cognitive load, excessively high levels of intrinsic and extraneous load can detrimentally impact the learning process. Therefore, the observed low germane loads in the initial week, characterized by increased intrinsic and extraneous load levels, were an anticipated outcome (Chandler & Sweller, 1996).

The research findings revealed an unexpected inconsistency, with the intrinsic and extraneous cognitive loads associated with the topic of variables being among the

lowest, while the germane load was notably the highest. This outcome was unexpected, particularly given that qualitative data indicated students frequently mentioned challenges in understanding the concept of variables ($f = 24$). Despite these reported difficulties, the corresponding cognitive load measurements did not align. In a study conducted with 6th, 7th, and 8th-grade students, Grover and Basu (2017) noted that students were unfamiliar with variable usage and held misconceptions, consistent with the findings of this research. In the present study, students reported finding the examples provided by the teacher during the explanation of the topic confusing and inconsistent. Additionally, they expressed difficulty in understanding how to use the variable code blocks, which differed slightly from what they had used previously in the digital programming environment.

The concept of variables constitutes a fundamental element of programming, yet it has been recognized as a challenging topic to both learn and teach (Dale, 2006; Holland et al., 1997). It is often perceived as abstract and challenging for novice programmers (Kohn, 2017). Studies have reported that students encounter difficulties in various aspects of variable usage, including establishing appropriate variable names, selecting suitable data types, distinguishing between mathematical symbols and programming operators, and correctly applying assignment and comparison operators (Mohamad Gobil et al., 2009). The lower intrinsic cognitive load observed in the current study regarding variables might be attributed to the less complex implementation of variables within the block-based programming environment employed. Since students were working with pre-defined code blocks, there was no requirement for them to explicitly define variables, specify data types, or assign values according to the data types. Similarly, the use of dropdown menus for selecting relational operators likely minimized the possibility of errors associated with these operators. Furthermore, it is also possible that students' prior exposure to variables within the context of loops, nested loops, and conditionals, without explicit instruction on variables, contributed to their apparent ease with this concept. However, qualitative data revealed a substantial knowledge gap regarding the fundamental nature and operation of variables despite their ability to complete tasks

by following procedural steps. While block-based environments alleviate the syntactic challenges associated with programming, they do not necessarily mitigate the conceptual difficulties inherent in understanding core programming constructs such as variables and loops. As Grover and Basu (2017) noted, students often struggle with grasping the essence of variables. In fact, the ability to successfully manipulate code blocks without a deep understanding of the underlying variable concepts might create an illusion of proficiency. There are studies in the literature that report contrary findings. For instance, in alignment with the quantitative results of this study, Grandell et al. (2006) found that variables were among the least challenging topics in their study conducted with high school students utilizing a text-based programming language. They also suggested that the deviation from the general understanding in the literature could be attributed, in part, to the type of programming language used in their study.

5.1.2 Gender

This study explored the potential gender disparities in middle programming education. The findings of this study revealed no statistically significant gender differences in programming education. Boys and girls displayed similar attitudes towards coding, had similar goal orientations and levels of self-efficacy, perceived similar classroom goal structures, used similar academic self-handicapping strategies, exhibited similar cheating behaviors, experienced similar cognitive load while learning to program, and ultimately achieved similar results on the programming achievement tests.

In contrast to the present study's findings, prior research has frequently documented a gender gap in programming education, with boys generally showing higher levels of interest, confidence, and performance in coding activities and technology-related careers compared to girls (e.g., Bergin & Reilly, 2006; Beyer et al., 2003; Cheryan et al., 2015; Doubé & Lang, 2012; Guzdia et al., 2014). These studies also explored the interaction of self-efficacy, intrinsic and extrinsic goal orientations,

programming success, and metacognitive strategies, finding that these factors impact student performance differently for males and females (Lishinski et al., 2016). Besides that, gender differences in programming beliefs were identified, where boys were more inclined towards computational thinking and saw programming as practical, while girls perceived programming as a creative and communicative activity (Tellhed et al., 2022), indicating that different aspects of programming appeal to boys and girls. The data from related studies suggested that these disparities were influenced by a combination of sociocultural factors, gender stereotypes, beliefs, the availability of role models, interest, computing self-efficacy, and prior experiences (Beyer, 2014; Cheryan et al., 2015; Doubé & Lang, 2012).

Although the gender gap in programming learning is a common narrative in literature, studies on gender differences have also shown mixed results. In the literature, there is a considerable number of studies that align with the findings of this research. For example, in the study by Kong et al. (2018), no significant difference was observed in the programming self-efficacy of the young learners despite the lower interest of the girls. Doubé & Lang (2012) also found similar results to those in this study, indicating no gender differences in terms of how much students valued computer programming, and both boys and girls seemed equally driven by a combination of wanting to achieve good grades or recognition and an interest in the subject matter. Additionally, there are studies in which no gender differences have been found in programming achievement scores, as in this study (Akinola, 2015; Bennedsen & Caspersen, 2005). Similarly, Qian & Lehman (2016) emphasized that differences in programming performance among middle school students are better explained by non-programming subjects rather than by gender. This consistency suggests that the lack of observed gender differences in programming education may be a more general phenomenon, not restricted to the specific context of this study.

Considering these previous research studies, several factors related to the absence of statistically significant differences in the investigated programming education variables in this study can be discussed. Studies suggest that when a field aligns with traditionally masculine traits in a specific culture (e.g., social isolation, intense focus

on technology, and innate brilliance), females tend to exhibit lower interest compared to males (Cheryan et al., 2015). These stereotypes and the gender construction of the discipline might explain the lack of adequate representation of females in disciplines such as computer science and engineering (Doubé & Lang, 2012). However, in research studies, it was emphasized that altering cultural perceptions and stereotypes surrounding computer science and engineering can positively influence girls' engagement and participation in these fields. Additionally, the role of media in shaping such stereotypes was highlighted, suggesting that media representations contribute to how girls perceive computer science courses and environments, ultimately impacting their interest in these fields (Cheryan et al., 2015). In the current study, given the unique characteristics of the study region, including its small-town setting and the fact that nearly half of the students reside in rural areas, the absence of observed gender differences compared to previous research suggests that media exposure may play a significant role in shaping students' perceptions and behaviors. This has arguably led to a shift in female students' perspectives on technology, potentially diminishing the previously observed gender gap reported in previous studies. Furthermore, there has been increased emphasis on coding education globally, including in Türkiye, and efforts have been made at the middle school level to promote this education. In this context, programming-related topics within the ITS curriculum were revised, and initiatives like the KodlaRize project championed programming education across all schools in the study's province. Additionally, there was an increased emphasis on integrating coding skills into classroom learning through various projects and technological equipment support for schools, as well as the establishment of coding centers. These initiatives likely contributed to a more standardized approach to programming education nationwide and within the study's province, potentially explaining the absence of significant variations in the investigated factors. Moreover, the increased emphasis on programming education has fostered a broader societal awareness of its importance. This is evident in the greater encouragement observed for girls to pursue computer-related fields compared to previous generations (Wang et al., 2015).

This study employed a block-based programming environment to introduce fundamental programming concepts to middle school students. Given the impact of block-based programming environments on students' attitudes towards programming, it can be inferred that the use of these environments might have contributed to the absence of a gender difference (Gunbatar & Karalar, 2018). However, the relatively low complexity of the programming tasks due to the circumscribed nature of block-based environments may have limited the potential for observing significant gender differences, which are more apparent in more complex programming contexts. Furthermore, the focus on introductory concepts might not fully capture the challenges associated with more advanced programming topics (Sullivan & Bers, 2016).

5.1.3 Geographical School Location

This study investigated the effects of geographical school location on programming success, math success, reading comprehension success, and various motivational constructs, including goal orientations, classroom goal structures, academic efficacy, cheating behavior, self-handicapping strategies, and attitudes toward programming education. The research involved students from three schools: one located in a central urban area with higher-income parents and two located in suburban areas with lower-income families and smaller student populations. The findings revealed significant differences between these groups, highlighting the importance of school location as a predictor of programming success and its impact on related motivational factors.

According to the results of this study, geographical school location was a strong predictor of the programming success of fifth-grade students. This finding aligns with previous research indicating that students from urban schools tend to achieve higher academic success compared to their suburban or rural counterparts (Bonilla-Mejía & Londoño-Ortega, 2021; Chand & Mohan, 2019; Panizzon, 2015). The reasons for this discrepancy include differences in both school and student

characteristics (Cresswell & Underwood, 2004). Urban schools generally provide better access to resources, better student exposure to technology, more experienced teachers, and a more stimulating educational environment, and are typically attended by students from higher socioeconomic backgrounds (Akpomudjere, 2020; Bouck, 2005; Chand & Mohan, 2019). Conversely, suburban and rural schools often face challenges in providing the same level of resources and support.

Socioeconomic factors are seen as a significant cause of the gap between schools from different geographical locations (Panizzon, 2015). Program for International Student Assessment (PISA) test results consistently demonstrate a strong correlation between socioeconomic status and student performance. PISA 2022 data aligns with the finding of this study, showing a gap of 82 points in mathematics scores between socio-economically advantaged and disadvantaged students from Türkiye (OECD, n.d.). Hanushek & Woessmann's (2012) research provided additional evidence that student and family background significantly impact educational outcomes. This aligns with the discussed point about the strong association between socioeconomic status and PISA results. Although this study did not directly investigate the impact of family factors as subfactors of geographical school location, considering the characteristics of the participants, it is evident that the family backgrounds of students in urban and suburban areas likely differed in terms of both education level and income status (see p. 51). Based on relevant literature, for instance, research by Marks et al. (2006), it could be suggested that these family background differences might have influenced the study's outcomes.

When examining the phenomenon within the specific context of programming learning, the importance of prior computing experience in programming success is particularly noteworthy (Grover et al., 2016; Zingaro, 2014). In this study, it was noted that students do not receive any formal computing-related instruction as part of the curriculum until the fifth grade. This means prior computing experience relies heavily on parental awareness and support. Parents with higher education levels and greater financial resources can provide more advantages for their children. This may include guiding them toward computing opportunities or providing more exposure

to technology, both of which can facilitate the development of computing skills at an early age (Metin et al., 2023). Additionally, related literature showed that the availability and quality of technological resources in schools also significantly impact programming success (Salleh Hudin, 2023). All the schools involved in the study had computer laboratories; however, the suburban schools faced challenges with older and fewer computers, as students mentioned in the interviews. During the study, a new computer lab was installed in one of the suburban schools as part of a project, but this improvement only occurred towards the end of the study period and did not significantly affect the outcomes.

In addition to programming success, geographical school location significantly influenced various motivational constructs. The study found significant differences in goal orientations and classroom goal structures between the two geographical locations. Urban school students exhibited higher performance-approach goal orientations and more favorable perceptions of classroom goal structures. This is consistent with previous research indicating that urban schools, with their higher resources and better-trained teachers, can create a more achievement-oriented environment that encourages students to set and pursue higher academic goals (Sun et al., 2022).

5.1.4 Mathematics Skills

One of the conclusions drawn from this research is that, among the various factors examined for success in computer programming, the most significant predictor was proficiency in mathematics. Given that the roots of computer programming lie in mathematics, a strong relationship between coding and mathematics is an expected outcome.

The emphasis on the strong positive correlation between students' mathematical abilities and their performance in introductory computer programming courses suggests that mathematical skills are crucial for understanding programming

concepts and logic. The relationship between mathematical and coding success is explored across various dimensions in the literature and is evidenced by multiple studies. Investigations into the relationship between mathematics achievement and programming performance have been conducted across various educational levels, spanning from primary and middle school (Bozal & Şendurur, 2024; Brannon & Novak, 2019; Calder, 2010; Grover et al., 2015, 2016; Hu et al., 2018; Qian & Lehman, 2016; Salac et al., 2021) to high school (Bennedsen & Caspersen, 2005; Erdogan et al., 2008; Nasution et al., 2022) and undergraduate or graduate (Baist & Pamungkas, 2017; Bergin & Reilly, 2006; Bubnic et al., 2024). For instance, Mathews (2017) emphasized the predictive power of prior mathematics performance on success in learning programming. Their study highlights that average mathematics grades from the previous year could strongly indicate of a student's ability to grasp programming concepts. Erdogan et al. (2008) conducted a study with high school students and found a significant relationship between mathematics achievement and programming achievement, although they did not find mathematics achievement to be a predictor of programming achievement. Grover et al. (2015) employed a design-based research approach to investigate students aged eleven to fourteen. Notably, their study yielded significant results, demonstrating that mathematical ability, alongside prior computing experience, serves as a highly strong predictor of successful learning outcomes in programming. The authors of the aforementioned study attributed this finding to the inherent nature of the assessments used to evaluate programming performance. These assessments frequently necessitate the application of mathematical knowledge. These findings directly align with the current study's emphasis on the critical role that a strong foundation in mathematical concepts plays in effectively completing coding tasks.

Similarly, Bennedsen & Caspersen (2005) investigated potential factors influencing success in an introductory programming course. Their analysis revealed that only two of the eight indicators were statistically significant. Mathematics grades from high school stood out as one of these critical predictors, explaining over 15% of the variation observed in exam grades. Bergin & Reilly (2006) corroborated the finding

that mathematics achievement strongly predicts performance in introductory programming courses. Bubnic et al. (2024) found that students with strong, complex problem-solving skills tended to perform better in introductory programming courses. The structural equation modeling results revealed that 64% of the variance in programming performance can be attributed to complex problem-solving skills. Similarly, Nasution et al. (2022) conducted a study in high school and found a positive correlation between the problem-solving abilities of students and their achievements in programming assignments.

Upon reviewing the literature, it becomes evident that several studies have identified mathematics as a significantly stronger predictor compared to the findings presented in this research. These results may stem from the investigation's focus on a lower grade level. Additionally, the study's measurement of programming knowledge and understanding among fifth graders who were newly introduced to computer science could contribute to this observation. This aligns with prior research that has established a moderate predictive role of mathematics in programming performance for younger students (e.g., Bennedsen & Caspersen, 2005).

The qualitative part of this research shed light on the specific difficulties students faced and the underlying factors that influenced their success in programming. According to the research findings, not only skills but also prior knowledge in mathematics emerged as a significant factor influencing programming success. Interviews with students revealed that mathematical concepts frequently posed difficulties when solving problems in the programming environment they used in their lessons. For instance, students often mentioned struggling with puzzles that required rotating characters at specific angles, indicating a lack of understanding of which angles to use. The qualitative analysis of this study goes beyond the initial finding of student difficulty with angles. Student interviews revealed challenges with other foundational mathematical concepts as well. These include directionality, the ability to understand and represent movement along a designated path, which is crucial for tasks involving directional commands within code. Spatial reasoning, the cognitive skills necessary to manipulate and understand objects in a spatial context,

also proved challenging. This is particularly relevant for tasks requiring the manipulation of on-screen objects or characters within a programmed environment. Finally, the framework for describing and locating points within a two-dimensional space, or coordinate systems, presented difficulties for students.

These findings highlight the importance of a strong foundation in these mathematical concepts for successful programming, as students struggled with specific coding examples that required applying these specific concepts. Calder (2010) employed a block-based programming environment that demonstrably fostered deeper engagement with geometric and measurement concepts. His study found that students readily grasped concepts of positionality, measurement (including coordinates, angles, and length), and spatial awareness within this environment. This aligns with the current study's findings, suggesting a potential link between a robust foundation in these mathematical concepts and success in programming tasks. Furthermore, Brannon & Novak (2019) directly corroborates this connection. Their investigation revealed that students encountering difficulties with mathematical content on the coding platform used in this study also exhibited struggles with geometric shapes, measurement, angles, and the coordinate system.

In a newly study, Bozal & Şendurur (2024) found no significant difference in computational thinking test scores between elementary school students who learned programming with math-supported activities and those who learned traditionally. The researchers explained the study's unexpected results in two ways. Firstly, the basic sorting tasks involved very beginner-level coding commands like moving and turning. These tasks likely didn't require advanced mathematical thinking, leading to similar scores across both groups. Secondly, the authors pointed out that current teaching methods might not be effective in truly merging mathematics and introductory computer science education. As the students' programming success in the current study was also assessed based on fundamental programming concepts, it is believed that the second factor (limitations in teaching methods) mentioned in the aforementioned study likely explains the absence of significant differences. These findings indicated that it is essential to ensure that computer science curricula

acknowledge the close relationship with mathematics. This underscores the need for improved pedagogical approaches to leverage the connection between mathematics and introductory computer science education for better learning outcomes in lower grades.

Teachers also play a vital role in this process. Preliminary research within this study showed that teachers, especially at the introductory level (such as fifth grade), frequently utilize learning platforms focused on coding, with code.org being a prominent example. Students typically engage in individual learning on these platforms in a computer lab setting, where teachers provide support by moving around the classroom. While teachers strive to assist students during individual computer-based learning, the number of students and time constraints can hinder their ability to provide adequate support and feedback, as emphasized by the interviewed students. Therefore, the appropriateness of tasks to students' readiness levels is paramount in such learning approaches. Teachers should carefully consider the mathematical connections of selected topics and examples when structuring lesson content, considering students' mathematical preparedness. Therefore, within the trend of teaching coding to all children, it is essential that students first build a strong foundation in mathematics to succeed in learning programming.

5.1.5 Reading Comprehension Skills

According to the findings of this study, reading comprehension achievement emerged as the second strongest predictor of academic success among middle school students. Given the nature of programming languages as high-level languages, which demand the skill to interpret meaning beyond literal statements and recognize patterns, reading proficiency is considered a fundamental prerequisite for effectively learning and utilizing a programming language. Additionally, these languages require the ability to synthesize information from code segments that might not be presented in a sequence and build mental models of abstract concepts (Salac et al., 2021; Schoeman, 2019). The relationship between reading comprehension ability

and programming success is multifaceted and significant in the related literature. While this study utilized pre-built code blocks, the findings aligned with existing research emphasizing the crucial role of code comprehension in successful programming.

There is a growing body of study that established reading comprehension as a significant predictor of programming success, underscoring the crucial role of effective code comprehension in programming proficiency. Lopez et al. (2008) investigated the relationship between code reading and code writing skills in novice programmers. They analyzed student performance on exam questions that involved code reading, tracing, and writing. Their findings showed a strong positive correlation between these skills, with code reading skills explaining 31% of the variation in student performance on code writing tasks. Similarly, Qian & Lehman (2016) conducted a study on Chinese students who were not native English speakers and found that proficiency in English was the strongest predictor of achievement in introductory programming. In this study, while mathematic ability also showed a correlation with performance, English proficiency emerged as the most significant factor. Grover et al. (2016) further supported this notion by demonstrating that, alongside math achievement, English ability served as a predictor of programming outcomes. These findings are consistent with the current study's emphasis on identifying key predictors of programming performance.

Reading comprehension is critically important for students to make sense of the code examples presented to them. This skill forms the foundation of programming learning by enabling them to decode the concepts, relationships, and logic within the code. As Lister et al. (2004) pointed out, students must be able to understand and analyze code examples to learn programming concepts effectively. In the current study, students did not have access to textbooks or printed materials. The absence of these materials in this block-based programming environment presented a unique challenge. While traditional classrooms might rely on students independently understanding code examples from textbooks, this was not an option in this study setting. Therefore, teachers became even more crucial in providing and explaining

code examples before transitioning students to practical exercises. This teacher-led approach ensures that students grasp the concepts and are prepared to apply their knowledge in real-world programming scenarios. Students' feedback during interviews aligns with this notion. When discussing unplugged activities (activities without computers), students mentioned the importance of understanding the concepts before moving on to practical exercises when answering related questions. This situation underscores the critical role of reading comprehension in block-based programming. Students could independently analyze and understand code examples by fostering reading comprehension skills, ultimately laying the foundation for successful application in practical exercises.

The literature also emphasizes the importance of reading comprehension, particularly in debugging. Reading code goes beyond just skimming it; it involves genuinely understanding what the code does. This allows programmers to identify and fix errors more easily (Perkins & Martin, 1986). While pre-built code blocks were used instead of traditional text-based programming in this study, the findings emphasized a similar relationship to text-based code reading. This suggests that code reading and comprehension skills are essential for debugging and overall code writing in programming, regardless of whether text-based or block-based.

Beyond traditional research methods, recent studies explore the link between reading comprehension and programming proficiency through eye-tracking and brainwave data. These studies reveal a fascinating connection: successful programmers exhibit distinct eye movement patterns and brain activity patterns. For instance, research by Ishida et al. (2020) and Ishida and Uwano (2019) suggested skilled programmers can rapidly shift their focus between problem specifications and the actual code. Additionally, their brainwaves showed an increase in specific frequencies over time, indicating heightened mental engagement. Further evidence comes from longitudinal eye-tracking studies by Andrzejewska and Kotoniak (2020). Their findings show that as students' programming skills improve, their eye movements become more efficient. This translates to increased distance traveled between fixations (saccade amplitude) and shorter fixation durations. These findings go beyond traditional text-

based assessments, suggesting that eye-tracking and brainwave data can offer valuable information about the cognitive processes underlying successful programming. This research not only supports the link between reading comprehension and programming but also highlights the value of alternative measurement methods.

While the majority of research underscores the positive influence of reading comprehension on programming proficiency, the bidirectional nature of this relationship has also been explored. Studies have demonstrated that computer instruction can enhance mathematical skills but may have less consistent effects on reading comprehension (Salac et al., 2021). Additionally, the impact of programming instruction on reading skills, as evidenced by Papatga and Ersoy (2016), suggests a potential complementary relationship between these two domains.

5.1.6 Attitude Toward Programming

This study identified attitude as a significant predictor of programming success. The research employed in-depth student interviews to gain a richer understanding of the factors influencing these attitudes. These interviews explored both positive and negative student perceptions of programming. In a qualitative analysis, the attitude was understood in a broader sense, reflecting an individual's expressed preferences and feelings toward engaging in a particular behavior (Fishman et al., 2021). This approach complements the investigation of other psychosocial constructs, such as goal orientation and self-efficacy, explored in this study.

Researchers have consistently identified attitude as a critical factor influencing student achievement. This holds true across various educational settings, including the field of computer science education. While a significant body of research has explored the attitudes of older students toward programming, investigations into the attitudes of younger learners are gaining increasing attention, and studies directly examining the impact of attitude on computer programming achievement are limited

(e.g., Deniz & Korucu, 2023; Love, 2023; Sun et al., 2022). Early studies primarily focused on the direct link between attitude and programming learning. More recent investigations have expanded the scope to examine the association between attitude and computational thinking, as well as its role in STEM education. It is noteworthy that a significant portion of the existing research aligns with the findings of this study, further reinforcing the notion that attitude plays a pivotal role in shaping student outcomes in computer science education (Sun et al., 2022).

In the literature, mathematics attitude has been identified as a factor positively influencing the computer programming learning of K-12 students (Ching et al., 2019; Ober et al., 2024). While a direct relationship between mathematics attitude and programming achievement was not explicitly tested in this study, the significant predictive power of mathematics achievement test scores for coding performance suggests that mathematics attitude may also play a positive role. This interpretation is supported by the established positive correlation between mathematics attitude and mathematics achievement in this study.

This study employed a combination of plugged and unplugged programming activities, with unplugged activities serving as an introduction to the concepts and plugged activities involving programming tasks on the code.org digital coding platform within a computer lab. Qualitative analysis revealed that students generally expressed more positive attitudes towards the plugged activities. In contrast to this study's finding, Love's (2023) study revealed a significant impact of physical computing activities on five attitude constructs among students: "definition, comfort, interest, classroom applications, and career/future use". Additionally, in this study, it was reported that 77% of the students expressed a preference for these physical activities over screen-based programming instruction. It is crucial to distinguish the physical activities employed in Love's study from the unplugged activities discussed in the present research. The former involved the interactive physical systems or devices that students program using software to create user-driven responses and behaviors, while the latter encompassed paper-based activities, games, and other unplugged experiences devoid of any integrated systems. In addition to the distinct

nature of the physical activities employed, several other factors could have contributed to the contrasting findings between Love's study and the present research. One potential explanation lies in the methodological approach. Love's study utilized a purely physical computing approach, while the present research utilized a mixed-methods approach that incorporated both unplugged and plugged (computer-based) programming activities. This difference in instructional strategies could have influenced student engagement and attitude formation. Moreover, the relative weight of unplugged and computer-based activities could have played a role in shaping the results. A greater emphasis on plugged activities might have resonated more strongly with students' desire for hands-on learning and potentially led to more positive attitudes. Additionally, considering the growing interest in technology among students, the present study's findings are not entirely surprising.

The results revealed a generally positive attitude towards programming. Students expressed a stronger preference for computer-based activities compared to unplugged activities that do not involve computers. Students were frequently observed to characterize this learning platform as enjoyable. This aligns with existing research suggesting a positive correlation between students' positive attitudes toward computers and their willingness to engage with programming. These results underscore the significance of considering student preferences when designing and developing programming education.

A notable finding from this study is the positive perception of enjoyment expressed by participants towards the unplugged activities. This aligns with previous research, such as Taub et al. (2012), where students consistently reported positive attitudes towards unplugged activities, often characterizing them as "fun" and engaging. In this study, while most of the students expressed a preference for computer-based (plugged) activities, they also highlighted positive aspects of unplugged activities, particularly emphasizing the value of social interaction in these settings, which was coded as "enjoyment of social interaction".

A separate study investigating the impact of pair programming on programming learning outcomes and attitudes revealed that attitudinal factors did not exert a significant influence on student learning within the pair programming setting (Vandenberg et al., 2021). This could be attributed to the collaborative nature of pair programming, where the shared learning environment and active engagement with a partner may mitigate the influence of individual attitudinal factors.

5.1.7 Patterns of Adaptive Learning

Among the subscales of the PALS, only the academic self-handicapping strategies variable emerged as a predictor of programming success in this study. The relationship between achievement and handicapping strategies was found to be negative. Although there has been limited research on self-handicapping strategies in programming, particularly in middle school contexts, studies in other domains have produced varying results regarding its relationship with achievement (Schwinger et al., 2014). However, the general trend, parallel to the findings of this study, suggests a negative correlation between self-handicapping and academic achievement (Urdu, 2004; Urdu et al., 1998). Schwinger et al. (2014) emphasized the influence of school type on this relationship. The finding that handicapping strategies emerged as a significant predictor in this study could be attributed, in part, to the school level, aligning with previous research suggesting a stronger association between self-handicapping and achievement in elementary schools compared to high schools (Leondari & Gonida, 2007).

Given the assumption that contextual and motivational factors can shape students' attitudes and behaviors, it is reasonable to expect disparities in academic self-handicapping strategies between urban and suburban school environments (Urdu & Midgley, 2001). While personal goal orientations and perceived classroom goal structures did not directly and significantly influence programming achievement in this study, the observed differences between urban and suburban schools in these variables suggest a more complex interplay between individual and contextual

factors that could potentially moderate the relationship with self-handicapping. Although the specific relationships between goal orientations and perceived goal structures with other variables were beyond the scope of this study, the literature points to a positive association between self-handicapping strategies and performance-avoid goals and classroom performance goal structure (Leondari & Gonida, 2007; Urdan, 2004) and a negative association with mastery goals (Midgley & Urdan, 2001).

The qualitative data from this study indicated a higher frequency of expressions related to mastery goal orientations among the students, such as career-oriented goals, challenge seeking, and relevance to daily life. However, there were also a notable number of expressions related to performance approach and performance-avoidance goals. Specifically, codes such as competition focus ($f = 4$), completion-driven motivation ($f = 11$), and fear of failure ($f = 6$) were identified as significant in the context of academic self-handicapping. The instructional environment used for the plugged activities in this study was game-based, where students progressed to the next level by completing puzzles designed to teach programming concepts. This type of performance-focused instructional practice has been reported to increase perceived classroom performance goals, which in turn can predict the use of self-handicapping strategies (Urdan et al., 1998). The emphasis on completing tasks to advance in levels may inadvertently encourage students to adopt self-handicapping behaviors to protect their self-esteem and mitigate fear of failure.

5.2 Conclusion

Programming education has increasingly become an essential skill and field to be introduced at various educational levels, including early childhood. However, as extensively discussed in the literature, students often face difficulties when learning programming. This study, utilizing a mixed-method design, investigated the computer programming learning processes of fifth-grade students who are new to coding and even computer science over a ten-week period. The study evaluated the

effects of sociodemographic attributes, educational background, affective and motivational learner characteristics, attitudes toward programming, and cognitive load levels of students on their programming learning.

A total of 199 students from three different schools participated in the study, with one school located in an urban area and the other two in suburban areas. Five research questions were addressed within the scope of this study. The first research question examined the changes in different types of cognitive load experienced by students while learning seven different coding topics (basic sequences, flowcharts, testing and debugging, loops, nested loops, flowcharts, variables). The second and third research questions investigated whether there were differences in the research variables based on students' gender and the geographical location of their schools. The fourth research question explored the extent to which the research variables explained changes in students' coding achievement. The final research question aimed to examine students' perspectives and experiences regarding the programming instruction process.

The study results indicated that students experienced high cognitive load, particularly with the concept of nested loops, due to its intrinsic complexity. On the other hand, it was unexpectedly found that students had difficulty with basic sequences in the first week. The interview findings revealed that the unplugged activity during the first week increased the students' intrinsic and extraneous cognitive load. Another example is the concept of variables. Although the intrinsic and extraneous loads for this topic were quite low, interviews indicated that students found the topic abstract and confusing. The examples provided during unplugged activities did not help in fully understanding this abstract concept. Furthermore, students reported that they did not fully understand how to use the relevant code block in a block-based programming environment that they applied to the same topic on the computer. This also highlights the importance of providing adequate pre-instructional guidance for self-regulated learning, especially since the sample consisted of students with no prior knowledge of computer science.

Unplugged activities were observed to be less preferred than computer-based activities, and students expressed less positive attitudes towards them. One of the reasons for this may be that some of these activities caused an increase in the cognitive load of the students.

Another factor that contributed to increased cognitive load and negatively affected the programming learning process was the students' lack of mathematical knowledge. Students particularly struggled with topics such as angles and the coordinate plane due to insufficient prior knowledge. Analysis results, consistent with the literature, also demonstrated that mathematical achievement is a significant predictor of coding success.

When examining whether gender characteristics developed a difference in the research variables, it was found that gender did not result in a significant difference for any of the variables. However, the geographical school location caused differences in both the affective and motivational variables, academic achievements, and the cognitive loads experienced by the students, all favoring urban schools. Students in suburban areas were typically of lower socioeconomic status and had less exposure to computers, significantly impacting their programming achievements and other programming-related variables. One unexpected finding was the lack of difference in attitudes between urban and suburban students. This can be attributed to several factors: the general fondness for computers among students, the teacher's efforts to create a positive classroom atmosphere, and the absence of programming questions in high-stakes exams like the high school entrance exams, students being exposed to ITS (Information Technology and Software) classes for only two years, which may contribute to more positive attitudes.

The study identified several variables that predict differences in students' programming achievement, including mathematics and reading comprehension performance, extraneous load, attitude, and academic self-handicapping strategies. The relationship between reading comprehension and programming learning has

been increasingly discussed in recent years, with a growing body of evidence supporting this connection.

In this study, the relationship was not influenced by the fact that programming languages are typically in English, as students created algorithms using a block-based application and selected the Turkish language option (their native language) on the website. This finding underscores the importance of language skills in understanding problem scenarios effectively, as programming requires an effective problem-solving approach.

Another significant variable closely related to coding success was extraneous load. In particular, poorly structured examples can increase extraneous load, especially for students with no prior experience or lower skills in mathematics and programming, thereby affecting their performance. Additionally, academic self-handicapping strategies emerged as an important predictor of coding success. The necessity of pair programming due to the lack of sufficient computers for each student led to some students not participating actively in the problem-solving process, as they relied on their partners. Interviews revealed that students who did not consider themselves successful often left the entire process to their partners and did not even look at the computer while their partners were solving the problems.

Given that a significant portion of the lessons involved computer-based activities, students who did not engage actively in these activities missed out on essential programming practice. Consequently, it is not surprising that academic self-handicapping strategies predict programming success. In pair programming, a student's passivity can also result from the dominance of the other student. This highlights the importance of carefully selecting pairs for pair programming and providing adequate guidance on how to engage effectively in this collaborative approach.

The large class size often limited the assistance teachers could provide during hands-on programming sessions, necessitating peer learning. While peer learning has benefits, such as promoting information sharing and supporting collaborative and

social learning, it also has drawbacks. For instance, dominant peers can overshadow others, some students might withdraw without challenging themselves, and peers might not always have the necessary knowledge or skills to explain concepts effectively to their peers. These issues sometimes led to students copying solutions from their peers. Furthermore, game-based learning activities, which were intended to engage students, also contributed to cheating behaviors. However, these behaviors were not found to be significant predictors of programming success. Although the relationship between game-based learning environments and students' performance-approach goal orientations was noted in interviews, these orientations, like other goal orientations and perceived classroom goal structures, were not significant predictors of coding success.

In summary, this study investigated the learning of fundamental programming concepts by fifth-grade students using a multifaceted approach. The findings revealed that students' academic backgrounds, specifically in mathematics and reading comprehension, were the most significant predictors of programming achievement. The study highlighted the difficulty in teaching concepts such as nested loops and variables in programming lessons. The importance of extraneous load in programming learning underscored the significance of instructional design. Among the affective and motivational factors, attitude and academic self-handicapping strategies were found to have a significant impact.

5.3 Implications of the Findings

Based on the findings of this study, the following recommendations are proposed for instructors and policymakers to effectively teach programming to middle school students who are novices in the subject:

- Introducing foundational skills before teaching coding can create a more positive and productive learning environment for students with no prior computer science experience. Initial lessons or activities aimed at

strengthening these foundational skills in computer science can help reduce feelings of frustration and overwhelm common among beginners. This approach can lead to a more sustained interest in programming, higher motivation, and a willingness to persevere through difficulties.

- Developing and maintaining an inclusive curriculum that highlights the relevance and application of programming skills to diverse fields is crucial. The relationship between programming and other fields can enhance students' learning experiences and outcomes.
- Mathematics is a critical foundation for programming success, making it essential to integrate mathematical considerations into computer science curricula. Incorporating mathematical principles such as algorithms, logic, data structures, and problem-solving techniques into programming education can strengthen students' mathematical skills and reinforce the connection between programming and mathematics. This approach can lead to improved learning outcomes and a deeper understanding of both subjects.
- Task design should align with students' mathematical readiness levels. It's important to be aware of students' mathematical backgrounds and provide additional support for those struggling with concepts like angles and coordinates, which can impact programming success. This approach ensures that all students, regardless of their initial proficiency, can engage with and succeed in programming tasks.
- Considering the role of reading comprehension in computer programming education is essential. Developing these skills helps students to understand the problem scenario, identify key information, and break down complex problems into manageable steps. This forms the foundation for writing efficient and accurate code.
- The findings suggest that interventions aiming to improve coding achievement should focus on reducing extraneous cognitive load. Paying close attention to activity design and examples can help minimize this load, especially for beginners. This might involve simplifying initial activities and

explanations or providing more scaffolding. Additionally, bridging the gap between unplugged activities and computer-based coding is crucial. Revisiting unplugged activities after the related coding concepts are learned may ensure a clear connection between unplugged activities and the programming concepts they introduce.

- The absence of textbooks limits the resources and materials available to students for their courses. If students also lack technological resources, they have no means to practice programming outside of school. Therefore, it is crucial to provide students with the necessary resources. Sharing these essential materials ensures that students can continue their learning and practice programming even outside the classroom, thereby supporting their educational development and success.
- Encouraging collaboration between teachers from different subjects might help create engaging and effective learning experiences that blend various concepts. By working together, teachers might develop lessons that integrate programming with other disciplines, making the learning process more dynamic and relevant. This interdisciplinary approach might also allow students to see the practical applications of programming in various fields and help them understand how knowledge from different areas interconnects, leading to a deeper and more comprehensive understanding of the material.
- Addressing the specific challenges faced by rural and suburban students highlights the need for a comprehensive approach. This includes improving educational resources in rural areas, supporting foundational academic skills, and creating an engaging learning environment that minimizes unnecessary cognitive load. Such efforts might help ensure that all students, regardless of their location, have access to high-quality education and opportunities for success in programming and other subjects.
- To optimize pair programming, educators should adopt a structured approach. This includes strategically pairing students based on skill, learning style, and personality to create a balanced learning environment. It is

important to provide clear guidelines for effective collaboration to prevent students from becoming passive learners. Monitoring and intervention strategies like targeted support and potential re-pairing might address imbalances and ensure all students benefit. Leveraging social learning dynamics through positive reinforcement, peer evaluation, and group discussions strengthens collaboration skills, might promote knowledge sharing, and fosters a deeper learning experience for all students.

- To foster positive attitudes towards coding, which might contribute to better learning outcomes, it is recommended to create an engaging and supportive learning environment. Integrating real-world applications of coding and highlighting its relevance across various fields might motivate students and enhance their interest in learning coding.
- While block-based programming environments are valuable for introducing younger students to coding, they can inadvertently create a competitive learning atmosphere through game-based elements. Structuring game-based learning activities around shared goals can help encourage students to work together towards a common objective. This fosters collaboration, communication, and problem-solving as a team rather than promoting individual competition. Such an approach helps to avoid outcomes that negatively impact students' learning and discourage self-handicapping behaviors.
- This study's findings showed that resource availability significantly affects programming education, suggesting that schools need to invest in up-to-date hardware and software to facilitate effective learning. Ensuring that students have access to the necessary technological tools is essential for providing a high-quality computer education and improving overall educational outcomes.

5.4 Recommendations for Further Research

This study investigated the variables associated with middle school students' success in learning fundamental computer programming concepts. To assess the model's generalizability, future research should explore its effectiveness in diverse educational settings, encompassing different schools, districts, or even countries. Additionally, researchers should employ a variety of programming languages and platforms to determine if the model applies equally well across diverse coding environments. The qualitative portion of this research identified other factors potentially influencing programming success that warrant further investigation. These include prior experiences with coding or technology, the effectiveness of paired programming compared to solo programming approaches, and the impact of technology exposure on learning outcomes. Notably, this study did not examine the changes in students' motivational factors throughout the educational process. Experimental studies could be designed to explore these changes and their potential relationship to programming success. It is also important to incorporate a wider range of assessment techniques to capture a more comprehensive picture of student learning. While this study did not directly link motivational variables to programming success, examining their relationships within the context of programming instruction for middle school students remains valuable. Furthermore, research should explore the effectiveness of different interventions in programming education to specifically address and potentially reduce the achievement gaps between students from diverse sociodemographic backgrounds. By pursuing these research avenues, future studies could contribute significantly to a deeper understanding of the factors that contribute to successful programming learning in middle school. This knowledge can then be used to develop more effective strategies for engaging and empowering all students in this critical field.

REFERENCES

- Abdul-Rahman, S. S., & Du Boulay, B. (2014). Learning programming via worked-examples: Relation of learning styles to cognitive load. *Computers in Human Behavior*, 30(1), 286–298. <https://doi.org/10.1016/j.chb.2013.09.007>
- Abdul Rahman, S. S. (2012). Learning programming via worked-examples: The effects of cognitive load and learning styles (Publication No. 1442498903.) [Doctoral dissertation, University of Sussex]. ProQuest Dissertations & Theses Global.
- Abou Naaj, M., & Nachouki, M. (2023). Students' perception of academic dishonesty in programming courses. *Journal of Further and Higher Education*, 47(1), 72–88. <https://doi.org/10.1080/0309877X.2022.2093630>
- Agnello, M. F., Araki, N., & Domenach, F. (2019). Building human infrastructure through programming and English Education in rural Japan. *International Journal for Talent Development and Creativity*, 7(1–2), 91–97. <https://files.eric.ed.gov/fulltext/EJ1297223.pdf>
- Akinola, S. O. (2015). Computer programming skill and gender difference: An empirical study. *American Journal of Scientific and Industrial Research*, 7(1), 1–6. <https://doi.org/10.5251/ajsir.2016.7.1.1.9>
- Akpomudjere, O. (2020). Effects of School location and teachers' quality on students performance in business studies examination in public secondary schools in Sapele local government area of Delta State. *Higher Education Studies*, 10(2), 114-121. <https://doi.org/10.5539/hes.v10n2p114>
- Albayrak, E., & Polat, E. (2022). Pair programming experiences of prospective information technologies teachers. *Bartın University Journal of Faculty of Education*, 11(2), 342–354. <https://doi.org/10.14686 / 991448>
- Altun, A., & Kasalak, İ. (2018). Blok temelli programlamaya (kodlama) ilişkin öz-yeterlik algısı ölçeği geliştirme çalışması. *Eğitim Teknolojisi Kuram ve Uygulama*, 8(1), 209–225. <https://doi.org/10.17943/etku.335916>

- Ames, C., & Archer, J. (1988). Achievement goals in the classroom: Students' learning strategies and motivation processes. *Journal of Educational Psychology*, 80(3), 260–267. <https://doi.org/10.1037//0022-0663.80.3.260>
- Anderman, E. M., & Midgley, C. (2004). Changes in self-reported academic cheating across the transition from middle school to high school. *Contemporary Educational Psychology*, 29(4), 499–517. <https://doi.org/10.1016/j.cedpsych.2004.02.002>
- Andrzejewska, M., & Kotoniak, P. (2020). Development of program comprehension skills by novice programmers – longitudinal eye tracking studies. *Informatics in Education*, 19(4), 521–541. <https://doi.org/10.15388/infedu.2020.23>
- Arevalo-Mercado, C. A., Munoz-Andrade, E. L., Cardona-Reyes, H., & Romero-Juarez, M. G. (2023). Applying cognitive load theory and the split attention effect to learning data structures. *Revista Iberoamericana de Tecnologías Del Aprendizaje*, 18(1), 107–113. <https://doi.org/10.1109/RITA.2023.3250580>
- Asad, K., Tibi, M., & Raiyn, J. (2016). Primary school pupils' attitudes toward learning programming through visual interactive environments. *World Journal of Education*, 6(5), 20–26. <https://doi.org/10.5430/wje.v6n5p20>
- Askar, P., & Davenport, D. (2009). An investigation of factors related to self-efficacy for java programming among engineering students. *Turkish Online Journal of Educational Technology*, 8(1), 26–32. <http://hdl.handle.net/11693/22504>
- Atkinson, R. K., Derry, S. J., Renkl, A., & Wortham, D. (2000). Learning from examples: Instructional principles from the worked examples research. *Review of educational research*, 70(2), 181–214. <https://doi.org/10.3102/00346543070002181>
- Ayre, C., & Scally, A. J. (2014). Critical values for Lawshe's content validity ratio: Revisiting the original methods of calculation. *Measurement and Evaluation in Counseling and Development*, 47(1), 79–86. <https://doi.org/10.1177/0748175613513808>
- Bæck, U. D. K. (2016). Rural location and academic success—remarks on research, contextualisation and methodology. *Scandinavian Journal of Educational Research*, 60(4), 435–448. <https://doi.org/10.1080/00313831.2015.1024163>

- Baist, A., & Pamungkas, A. S. (2017). Analysis of student difficulties in computer programming. *VOLT: Jurnal Ilmiah Pendidikan Teknik Elektro*, 2(2), 81. <https://doi.org/10.30870/volt.v2i2.2211>
- Bandura, A. (1977). Self-efficacy: Toward a unifying theory of behavioral change. *Psychological Review*, 84(2), 191–215. <https://doi.org/10.1037/0033-295X.84.2.191>
- Barros, B., Conejo, R., Ruiz-Sepulveda, A., & Triguero-Ruiz, F. (2021). I explain, you collaborate, he cheats: An empirical study with social network analysis of study groups in a computer programming subject. *Applied Sciences*, 11(19), 1–32. <https://doi.org/10.3390/app11199328>
- Baser, M. (2013). Attitude, gender and achievement in computer programming. *Middle East Journal of Scientific Research*, 14(2), 248–255. <https://doi.org/10.5829/idosi.mejsr.2013.14.2.2007>
- Bau, D., Gray, J., Kelleher, C., Sheldon, J., & Turbak, F. (2017). Learnable programming: Blocks and beyond. *Communications of the ACM*, 60(6), 72–80. <https://doi.org/10.1145/3015455>
- Bennedsen, J., & Caspersen, M. E. (2005). An investigation of potential success factors for an introductory model-driven programming course. *Proceedings of the 1st International Computing Education Research Workshop, ICER 2005*, 155–163. <https://doi.org/10.1145/1089786.1089801>
- Bergey, B. W., Ketelhut, D. J., Liang, S., Natarajan, U., & Karakus, M. (2015). Scientific inquiry self-efficacy and computer game self-efficacy as predictors and outcomes of middle school boys' and girls' performance in a science assessment in a virtual environment. *Journal of Science Education and Technology*, 24(5), 696–708. <https://doi.org/10.1007/s10956-015-9558-4>
- Bergin, S., & Reilly, R. (2006). Predicting introductory programming performance: A multi-institutional multivariate study. *Computer Science Education*, 16(4), 303–323. <https://doi.org/10.1080/08993400600997096>
- Bergman, E. M., De Bruin, A. B. H., Vorstenbosch, M. A. T. M., Kooloos, J. G. M., Puts, G. C. W. M., Leppink, J., Scherpbier, A. J. J. A., & Van Der Vleuten, C. P. M. (2015). Effects of learning content in context on knowledge acquisition

- and recall: A pretest-posttest control group design. *BMC Medical Education*, 15(1), 1–12. <https://doi.org/10.1186/s12909-015-0416-0>
- Berssanette, J. H., & De Francisco, A. C. (2022). Cognitive load theory in the context of teaching and learning computer programming: A systematic literature review. *IEEE Transactions on Education*, 65(3), 440–449. <https://doi.org/10.1109/TE.2021.3127215>
- Beyer, S. (2014). Why are women underrepresented in Computer Science? Gender differences in stereotypes, self-efficacy, values, and interests and predictors of future CS course-taking and grades. *Computer Science Education*, 24(2–3), 153–192. <https://doi.org/10.1080/08993408.2014.963363>
- Beyer, S., Rynes, K., Perrault, J., Hay, K., & Haller, S. (2003). Gender differences in computer science students. *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education, USA*, 34(1), 49–53. <https://doi.org/10.1145/611892.611930>
- Blanco, J., Losano, L., Aguirre, N., Novaira, M. M., Permigiani, S., & Scilingo, G. (2009). An introductory course on programming based on formal specification and program calculation. *SIGCSE Bulletin Inroads*, 41(2), 31–37. <https://doi.org/10.1145/1595453.1595459>
- Bliss, T. V. P., & Collingridge, G. L. (1993). A synaptic model of memory: Long-term potentiation in the hippocampus. *Nature*, 361(6407), 31–39. <https://doi.org/10.1038/361031a0>
- Bonilla-Mejía, L., & Londoño-Ortega, E. (2021). *Geographic isolation and learning in rural schools* (BDE Publication No. 1169). Borradores de Economía. <https://repositorio.banrep.gov.co/server/api/core/bitstreams/0a9e6cc6-89b0-4d3b-a255-03db6556b42f/content>
- Bouck, E. C. (2005). Service delivery and instructional programming in rural, suburban, and urban secondary special education: An exploratory study. *Rural Special Education Quarterly*, 24(4), 18–25. <https://doi.org/10.1177/875687050502400404>
- Bounajim, D., Rachmatullah, A., Hinckle, M., Mott, B., Lester, J., Smith, A., Emerson, A., Fahid, F. M., Tian, X., Wiggins, J. B., Boyer, K. E., & Wiebe, E.

- (2021). Applying cognitive load theory to examine STEM undergraduate students' experiences in an adaptive learning environment: A mixed-methods study. *Proceedings of the Human Factors and Ergonomics Society*, 65(1), 556–560. <https://doi.org/10.1177/1071181321651249>
- Bowman, N. A., Jarratt, L., Culver, K. C., & Segre, A. M. (2019). How prior programming experience affects students' pair programming experiences and outcomes. *Proceedings of the ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE, UK*, 24(1), 170–175. <https://doi.org/10.1145/3304221.3319781>
- Bozal, M., & Şendurur, P. (2024). The effect of introductory programming education on computational thinking. *Instructional Technnology and Lifelong Learning*, 5(1), 21–46. <https://doi.org/10.52911/ital.1394556>
- Brannon, M., & Novak, E. (2019). Coding success through math intervention in an elementary school in rural Amish Country. *Journal of Computer Science Integration*, 2(2). <https://doi.org/10.26716/jcsi.2019.02.2.1>
- Bruckman, A., Jensen, C., & Debonte, A. (2002). Gender and programming achievement in a CSCL environment. *Proceedings of the Conference on Computer Support for Collaborative Learning: Foundations for a CSCL Community, USA*, 4(1), 119-127. <https://dl.acm.org/doi/10.5555/1658616.1658634>
- Bubnic, B., Mernik, M., & Kosar, T. (2024). Exploring the predictive potential of complex problem-solving in computing education: A case study in the introductory programming course. *Mathematics*, 12(11), 1-27. <https://doi.org/10.3390/math12111655>
- Bucks, G., & Oakes, W. C. (2011). Phenomenography as a tool for investigating understanding of computing concepts. *Proceedings of ASEE Annual Conference and Exposition, Texas*, 3(1), 1-22. <https://doi.org/10.18260/1-2--18485>
- Çakiroğlu, Ü., Suiçmez, S. S., Kurtoğlu, Y. B., Sari, A., Yıldız, S., & Öztürk, M. (2018). Exploring perceived cognitive load in learning programming via scratch. *Research in Learning Technology*, 26(1), 1–20. <https://doi.org/10.25304/rlt.v26.1888>

- Calder, N. (2010). Using Scratch: An integrated problem-solving approach to mathematical thinking. *Australian Primary Mathematics Classroom*, 15(4), 9–14. <https://files.eric.ed.gov/fulltext/EJ906680.pdf>
- Caspersen, M. E., & Bennedsen, J. (2007). Instructional design of a programming course – A learning theoretic approach. *Proceedings of the 3rd International Workshop on Computing Education Research, USA*, 3(1), 111–122. <https://doi.org/10.1145/1288580.1288595>
- Cesur Özkara, E., & Yanpar Yelken, T. (2020). Ortaöğretim öğrencilerine yönelik programlama öz yeterlik ölçeğinin geliştirilmesi: Geçerlik ve güvenirlik çalışması. *Eğitim Teknolojisi Kuram ve Uygulama*, 10(2), 345–365. <https://doi.org/10.17943/etku.632606>
- Chand, D., & Mohan, P. (2019). Impact of school locality on teaching and learning: A qualitative inquiry. *Waikato Journal of Education*, 24(2), 65–72. <https://doi.org/10.15663/wje.v24i2.672>
- Chandler, P., & Sweller, J. (1994). Why some material is difficult to learn. *Cognition and Instruction*, 12(3)185–233. https://doi.org/10.1207/s1532690xc1203_1
- Chandler, P., & Sweller, J. (1996). Cognitive load while learning to use a computer program. *Applied Cognitive Psychology*, 10(2), 151–170. [https://doi.org/10.1002/\(SICI\)1099-0720\(199604\)10:2<151::AID-ACP380>3.0.CO;2-U](https://doi.org/10.1002/(SICI)1099-0720(199604)10:2<151::AID-ACP380>3.0.CO;2-U)
- Chase, W. G., & Simon, H. A. (1973). Perception in chess. *Cognitive Psychology*, 4(1), 55–81. [https://doi.org/10.1016/0010-0285\(73\)90004-2](https://doi.org/10.1016/0010-0285(73)90004-2)
- Cheryan, S., Master, A., & Meltzoff, A. N. (2015). Cultural stereotypes as gatekeepers: Increasing girls' interest in computer science and engineering by diversifying stereotypes. *Frontiers in Psychology*, 6(1), 1–8. <https://doi.org/10.3389/fpsyg.2015.00049>
- Chi, M. T. H., Glaser, R., & Rees, E. (1982). Expertise in problem solving. In R.J. Sternberg (Ed.), *Advances in the Psychology of Human Intelligence* (Vol. 1, pp. 7-75). Erlbaum
- Ching, Y. H., Yang, D., Wang, S., Baek, Y., Swanson, S., & Chittoori, B. (2019).

Elementary school student development of STEM attitudes and perceived learning in a STEM integrated robotics curriculum. *TechTrends*, 63(5), 590–601. <https://doi.org/10.1007/s11528-019-00388-0>

Code.org. (2022). *Code.org 2022 Annual Report*. <https://code.org/about/2022>

Code.org. (2024). *About us*. <https://code.org/about>

Cohen, L., Manion, L., & Morrison, K. (2017). *Research Methods in Education*. Routledge. <https://doi.org/10.4324/9781315456539>

Coleman, S. A., & Nichols, E. (2011). Embedding inquiry based learning into programming via paired assessment. *ITALICS Innovations in Teaching and Learning in Information and Computer Sciences*, 10(1), 72–77. <https://doi.org/10.11120/ital.2011.10010072>

Cooper, G. (1998). Research into cognitive load theory and instructional design at UNSW. *Cognitive Load Theory and Instructional Design at UNSW*. http://penta2.ufrgs.br/edu/edu3375/CLT_NET_Aug_97.HTML

Corbin, J., & Strauss, A. (2012). *Basics of qualitative research: Techniques and procedures for developing grounded theory* (4th ed.). SAGE.

Cowan, N. (2014). Working memory underpins cognitive development, learning, and education. *Educational Psychology Review*, 26(2), 197–223. <https://doi.org/10.1007/s10648-013-9246-y>

Cresswell, J., & Underwood, C. (2004). *Location, location, location: Implications of geographic situation on Australian student performance in PISA 2000* (ACER Research Monograph Publication No.58). https://research.acer.edu.au/acer_monographs/2

Creswell, J. W. (2007). *Qualitative Inquiry and Research Design*. SAGE

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

- Creswell, J. W. (2015). *Educational research: Planning, conducting, and evaluating quantitative and qualitative research* (5th ed.). Pearson Education.
- Creswell, J. W., & Miller, D. L. (2000). Determining validity in qualitative inquiry. *Theory Into Practice*, 39(3), 124–130. https://doi.org/10.1207/s15430421tip3903_2
- Dale, N. B. (2006). Most difficult topics in CS1. *ACM SIGCSE Bulletin*, 38(2), 49–53. <https://doi.org/10.1145/1138403.1138432>
- De Groot, A. D. (1978). *Thought and Choice in Chess* (2nd ed.). Walter De Gruyter
- de Vink, I. C., Tolboom, J. L. J., & van Beekun, O. (2023). Exploring the effects of near-peer teaching in ro-botics education: The role of STEM attitudes. *Informatics in Education*, 22(2), 329–350. <https://doi.org/10.15388/infedu.2023.10>
- Deniz, T., & Korucu, A. T. (2023). The effect of coding education designed with different visual programs on academic success and attitudes and self-efficiencies of secondary school students. *Journal of Teacher Education and Lifelong Learning*, 5(1), 307–323. <https://doi.org/10.51535/tell.1279547>
- Denzin, N. K. (2012). Triangulation 2.0*. *Journal of Mixed Methods Research*, 6(2), 80–88. <https://doi.org/10.1177/1558689812437186>
- Doubé, W., & Lang, C. (2012). Gender and stereotypes in motivation to study computer programming for careers in multimedia. *Computer Science Education*, 22(1), 63–78. <https://doi.org/10.1080/08993408.2012.666038>
- Duran, R., Zavgorodniaia, A., & Sorva, J. (2022). Cognitive Load theory in computing education research: A review. *ACM Transactions on Computing Education*, 22(4), 1–27. <https://doi.org/10.1145/3483843>
- Dweek, C. S. (1986). *Dweek (1986)*. Motivational processes affecting learning. *American Psychologist*, 41(10), 1040–1048. <https://doi.org/10.1037/0003-066X.41.10.1040>
- Elliot, A. J., & McGregor, H. A. (2001). A 2 × 2 achievement goal framework.

- Journal of Personality and Social Psychology*, 80(3), 501–519.
<https://doi.org/10.1037/0022-3514.80.3.501>
- Elliot, A. J., Murayama, K., & Pekrun, R. (2011). A 3×2 achievement goal model. *Journal of Educational Psychology*, 103(3), 632–648.
<https://doi.org/10.1037/a0023952>
- Elliott, E. S., & Dweck, C. S. (1988). Goals: An approach to motivation and achievement. *Journal of Personality and Social Psychology*, 54(1).
<https://doi.org/10.1037/0022-3514.54.1.5>
- Erdem, E. (2018). *Blok tabanlı ortamlarda programlama öğretimi sürecinde farklı öğretim stratejilerinin çeşitli değişkenler açısından incelenmesi*. [Master Thesis, Başkent Üniversitesi]. BU Repository.
<http://acikerisim.baskent.edu.tr:8080/handle/11727/2903>
- Erdogan, Y., Aydin, E., & Kabaca, T. (2008). Exploring the psychological predictors of programming achievement. *Journal of Instructional Psychology*, 35, 264–271. http://imo.pau.edu.tr/tolga/predictor_programming.pdf
- Erten, I. H. (2015). Validating Myself-As-A-Learner Scale (MALS) in the Turkish Context. *Novitas-ROYAL (Research on Youth and Language)*, 9(1), 46–59.
<https://files.eric.ed.gov/fulltext/EJ1167210.pdf>
- Field, A., Miles, J., & Field, Z. (2016). *Discovering statistics using R*. SAGE
- Field, A. (2005). *Discovering Statistics Using SPSS* (3rd ed.). SAGE.
- Fluck, A., Webb, M., Cox, M., Angeli, C., Malyn-Smith, J., Voogt, J., & Zagami, J. (2016). Arguing for computer science in the school curriculum. *Educational Technology & Society*, 19(3), 38–46.
<https://www.jstor.org/stable/jeductechsoci.19.3.38>
- Fraenkel, J. R., Wallen, N. E., & Hyun, H. H. (2012). *How to design and evaluate research in education* (8th ed.). McGraw-Hill.
- Fredericks, S., ElSayed, M., Hammad, M., Abumiddain, O., Istwani, L., Rabeea, A., Rashid-Doubell, F., & Bella, A. M. E. (2021). Anxiety is associated with

- extraneous cognitive load during teaching using high-fidelity clinical simulation. *Medical Education Online*, 26(1), 1-8. <https://doi.org/10.1080/10872981.2021.1994691>
- Gaddy, V., & Ortega, F. R. (2022). Exploring factors associated with retention in computer science using virtual reality. *Proceedings of the Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops, VRW*, New Zealand, 27(1), 271–276. <https://doi.org/10.1109/VRW55335.2022.00062>
- Garner, S. (2002). Reducing the cognitive load on novice programmers. In P. Barker & S. Rebelsky (Eds.), *Proceedings of World Conference on Educational Multimedia, Hypermedia & Telecommunications* (pp. 578-583). Denver, Colorado, USA: Association for the Advancement of Computing in Education (AACE). <https://www.learntechlib.org/p/10329>.
- Geary, D. C. (2008). An evolutionarily informed education science. *Educational Psychologist*, 43(4), 179–195. <https://doi.org/10.1080/00461520802392133>
- Geisinger, K. F. (1994). Cross-cultural normative assessment: Translation and adaptation issues influencing the normative interpretation of assessment instruments. *Psychological Assessment*, 6(4), 304–312. <https://doi.org/10.1037/1040-3590.6.4.304>
- Gerson, S. A., Morey, R. D., & van Schaik, J. E. (2022). Coding in the cot? Factors influencing 0–17s’ experiences with technology and coding in the United Kingdom. *Computers and Education*, 178(1), 1-16. <https://doi.org/10.1016/j.compedu.2021.104400>
- Gilhooly, K. J., & Green, A. J. K. (1988). The use of memory by experts and novices. *Advances in Psychology*, 55(1), 379-395. [https://doi.org/10.1016/S0166-4115\(08\)60635-4](https://doi.org/10.1016/S0166-4115(08)60635-4)
- Gomes, A., Ke, W., Lam, C. T., Teixeira, A. R., Correia, F. B., Marcelino, M. J., & Mendes, A. J. (2019). Understanding loops: a visual methodology. *International Conference on Engineering, Technology and Education, China*, 3(1), 1–7. <https://doi.org/10.1109/TALE48000.2019.9225951>
- Grandell, L., Peltomäki, M., Back, R. J., & Salakoski, T. (2006). Why complicate things? Introducing programming in high school using Python. *Proceedings of*

the 8th Australasian Conference on Computing Education, Australia, 52(1), 71–80. <https://dl.acm.org/doi/pdf/10.5555/1151869.1151880>

Gray, S., Clair, C. S., James, R., Park, W., & Mead, J. (2007). Suggestions for graduated exposure to programming concepts using fading worked examples. *Proceedings of the 3rd International Workshop on Computing Education Research, USA*, 3(1), 99–110. <https://doi.org/10.1145/1288580.1288594>

Greenberg, K., & Zheng, R. (2023). Revisiting the debate on germane cognitive load versus germane resources. *Journal of Cognitive Psychology*, 35(3), 295–314. <https://doi.org/10.1080/20445911.2022.2159416>

Greifenstein, L., Graßl, I., & Fraser, G. (2021). Challenging but full of opportunities: Teachers' perspectives on programming in primary schools. *ACM International Conference Proceeding Series, Finland*, 21(1), 1–10. <https://doi.org/10.1145/3488042.3488048>

Grover, S. (2020). Designing an assessment for introductory programming concepts in middle school computer science. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, 20(1) 678–684. <https://doi.org/10.1145/3328778.3366896>

Grover, S., & Basu, S. (2017). Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and Boolean logic. *Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE*, 267–272. <https://doi.org/10.1145/3017680.3017723>

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>

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, 25(2), 199–237. <https://doi.org/10.1080/08993408.2015.1033142>

Grover, S., Pea, R., & Cooper, S. (2016). Factors influencing computer science learning in middle school. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education, USA*, 47(1), 552–557.

<https://doi.org/10.1145/2839509.2844564>

- Guenaga, M., Eguíluz, A., Garaizar, P., & Gibaja, J. (2021). How do students develop computational thinking? Assessing early programmers in a maze-based online game. *Computer Science Education*, 31(2), 259–289. <https://doi.org/10.1080/08993408.2021.1903248>
- Gunbatar, M. S., & Karalar, H. (2018). Gender differences in middle school students' attitudes and self-efficacy perceptions towards MBlock programming. *European Journal of Educational Research*, 7(4), 925–933. <https://doi.org/10.12973/eu-jer.7.4.923>
- Guo, M., & Hu, X. (2022). Relationship of classroom goal structures to Chinese Miao and Han students' goal orientations and mathematics achievement. *Asia-Pacific Education Researcher*, 31(4), 345–355. <https://doi.org/10.1007/s40299-021-00576-8>
- Guzdial, M., Ericson, B., McKlin, T., & Engelman, S. (2014). Georgia computes! An intervention in a US state, with formal and informal education in a policy context. *ACM Transactions on Computing Education*, 14(2), 1–29. <https://doi.org/10.1145/2602488>
- Haden, P., Gasson, J., Wood, K., & Parsons, D. (2016). Can you learn to teach programming in two days? *ACM International Conference Proceeding Series*, 1(1), 1–7. <https://doi.org/10.1145/2843043.2843063>
- Hadie, S. N. H., & Yusoff, M. S. B. (2016). Assessing the validity of the cognitive load scale in a problem-based learning setting. *Journal of Taibah University Medical Sciences*, 11(3), 194–202. <https://doi.org/10.1016/j.jtumed.2016.04.001>
- Hanushek, E. A., & Woessmann, L. (2012). Do better schools lead to more growth? Cognitive skills, economic outcomes, and causation. *Journal of Economic Growth*, 17(4), 267–321. <https://doi.org/10.1007/s10887-012-9081-x>
- Harms, K. J. (2013). Applying cognitive load theory to generate effective programming tutorials. *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, USA*, 1(1), 179–180. <https://doi.org/10.1109/VLHCC.2013.6645274>

- Harms, K. J., Chen, J., & Kelleher, C. (2016). Distractors in parsons problems decrease learning efficiency for young novice programmers. *ICER 2016 - Proceedings of the 2016 ACM Conference on International Computing Education Research*, 241–250. <https://doi.org/10.1145/2960310.2960314>
- Hawi, N. (2010). Causal attributions of success and failure made by undergraduate students in an introductory-level computer programming course. *Computers and Education*, 54(4), 1127–1136. <https://doi.org/10.1016/j.compedu.2009.10.020>
- Hazley, M. P., Shell, D. F., Soh, L. K., Miller, L. D., Chiriacescu, V., & Ingraham, E. (2015). Changes in student goal orientation across the semester in undergraduate computer science courses. *Proceedings - Frontiers in Education Conference, FIE, USA*, 1(1), 1-7. <https://doi.org/10.1109/FIE.2014.7044366>
- Hellas, A., Leinonen, J., & Ihantola, P. (2017). Plagiarism in take-home exams: Help-seeking, collaboration, and systematic cheating. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE, Italy*, 22(1), 238–243. <https://doi.org/10.1145/3059009.3059065>
- Hinckle, M., Rachmatullah, A., Mott, B., Boyer, K. E., Lester, J., & Wiebe, E. (2020). The relationship of gender, experiential, and psychological factors to achievement in computer science. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE, Norway*, 25(1), 225–231. <https://doi.org/10.1145/3341525.3387403>
- Holland, S., Griffiths, R., & Woodman, M. (1997). Avoiding object misconceptions. *Proceedings of the 28th SIGCSE Technical Symposium on Computer Science Education, USA*, 29(1). 131–134. <https://doi.org/10.1145/268084.268132>
- Hsu, J. M., Chang, T. W., & Yu, P. T. (2012). Learning effectiveness and cognitive loads in instructional materials of programming language on single and dual screens. *Turkish Online Journal of Educational Technology*, 11(2), 156–166. <https://files.eric.ed.gov/fulltext/EJ989024.pdf>
- Hu, X., Gong, Y., Lai, C., & Leung, F. K. S. (2018). The relationship between ICT and student literacy in mathematics, reading, and science across 44 countries: A multilevel analysis. *Computers and Education*, 125(1), 1–13. <https://doi.org/10.1016/j.compedu.2018.05.021>

- Hulleman, C. S., Schrager, S. M., Bodmann, S. M., & Harackiewicz, J. M. (2010). A Meta-analytic review of achievement goal measures: Different labels for the same constructs or different constructs with similar labels? *Psychological Bulletin*, 136(3), 422–449. <https://doi.org/10.1037/a0018947>
- Ibanez, M. B., Di-Serio, A., & Delgado-Kloos, C. (2014). Gamification for engaging computer science students in learning activities: A case study. *IEEE Transactions on Learning Technologies*, 7(3), 291–301. <https://doi.org/10.1109/TLT.2014.2329293>
- Ishida, T., & Uwano, H. (2019). Synchronized analysis of eye movement and EEG during program comprehension. *Proceedings of International Workshop on Eye Movements in Programming, USA*, 6(1), 26–32. <https://doi.org/10.1109/EMIP.2019.00012>
- Ishida, T., Uwano, H., & Ikutani, Y. (2020). Combining biometric data with focused document types classifies a success of program comprehension. *IEEE International Conference on Program Comprehension, South Korea*, 28(1), 366–370. <https://doi.org/10.1145/3387904.3389291>
- Hair, J.F., Hult, G.T.M., Ringle, C.M., Sarstedt, M.(2021). *A primer on partial least squares structural equation modeling (PLS-SEM)* (3rd ed.).SAGE
- Kalyuga, S. (2011). Cognitive load theory: How many types of load does it really need? *Educational Psychology Review*, 23(1), 1–19. <https://doi.org/10.1007/s10648-010-9150-7>
- Kanaparan, G., Cullen, R., & Mason, D. (2017). Effect of self-efficacy and emotional engagement on introductory programming students. *Australasian Journal of Information Systems*, 23(1), 1–21. <https://doi.org/10.3127/ajis.v23i0.1825>
- Kandel, E. R., Dudai, Y., & Mayford, M. R. (2014). The molecular and systems biology of memory. *Cell*, 157(1), 163–186. <https://doi.org/10.1016/j.cell.2014.03.001>
- Karalar, H. (2023). Adaptation of computer programming self-efficacy scale for computer literacy education into Turkish for middle school students. *International Technology and Education Journal*, 7(2), 51–59. <http://itejournal.com/>

- Karaman, U., & Büyükalan Filiz, S. (2019). Kodlama eğitimine yönelik tutum ölçeği'nin (KEYTÖ) geliştirilmesi. *Gelecek Vizyonlar Dergisi*, 3(2), 36–47. <https://doi.org/10.29345/futvis.80>
- Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys*, 37(2), 83–137. <https://doi.org/10.1145/1089733.1089734>
- Ketenci, T., Calandra, B., Margulieux, L., & Cohen, J. (2019). The relationship between learner characteristics and student outcomes in a middle school computing course: An exploratory analysis using structural equation modeling. *Journal of Research on Technology in Education*, 51(1), 63–76. <https://doi.org/10.1080/15391523.2018.1553024>
- Kim, Y., Lee, K. & Park, H. (2022). Watcher : Cloud-based coding activity tracker for fair evaluation of programming assignments. *Sensors*, 22(19)1–18. <https://doi.org/10.3390/s22197284>
- Kinnunen, P., & Simon, B. (2011). CS majors' self-efficacy perceptions in CS1: Results in light of social cognitive theory. *Proceedings of the International Computing Education Research, USA*, 7(1), 19–26. <https://doi.org/10.1145/2016911.2016917>
- Kittur, J. (2020). Measuring the programming self-efficacy of electrical and electronics engineering students. *IEEE Transactions on Education*, 63(3), 216–223. <https://doi.org/10.1109/TE.2020.2975342>
- Kohn, T. (2017). Variable evaluation: An exploration of novice programmers' understanding and common misconceptions. *Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE, Italy*, 22(1), 345–350. <https://doi.org/10.1145/3017680.3017724>
- Kong, S. C., Chiu, M. M., & Lai, M. (2018). A study of primary school students' interest, collaboration attitude, and programming empowerment in computational thinking education. *Computers and Education*, 127(3), 178–189. <https://doi.org/10.1016/j.compedu.2018.08.026>
- Koray, A., & Bilgin, E. (2023). The effect of block coding (Scratch) activities

- integrated into the 5E Learning Model in science teaching on students' computational thinking skills and programming self-efficacy. *Science Insights Education Frontiers*, 18(1), 2825–2845. <https://doi.org/10.15354/sief.23.or410>
- Korkmaz, Ö., & Altun, H. (2014). Adapting computer programming self-efficacy scale and engineering students' self-efficacy perceptions. *Participatory Educational Research*, 1(1), 20–31. <https://doi.org/10.17275/per.14.02.1.1>
- Kovari, A., & Katona, J. (2023). Effect of software development course on programming self-efficacy. *Education and Information Technologies*, 28(9), 10937–10963. <https://doi.org/10.1007/s10639-023-11617-8>
- Krieglstein, F., Beege, M., Rey, G. D., Ginns, P., Krell, M., & Schneider, S. (2022). A systematic meta-analysis of the reliability and validity of subjective cognitive load questionnaires in experimental multimedia learning research. *Educational Psychology Review*, 34(4), 2485–2541. <https://doi.org/10.1007/s10648-022-09683-4>
- Kukul, V., Gökçearsan, Ş., & Günbatar, M. S. (2017). Computer Programming Self-Efficacy Scale (CPSES) for secondary school students: Development, validation and reability. *Eğitim Teknolojisi Kuram Ve Uygulama*, 7(1), 158–179. <https://doi.org/10.17943/etku.288493>
- Kurasaki, K. S. (2000). Intercoder reliability for validating conclusions drawn from open-ended interview data. *Field Methods*, 12(3), 179–194. <https://doi.org/10.1177/1525822X0001200301>
- Lambić, D., Đorić, B., & Ivakić, S. (2021). Investigating the effect of the use of code.org on younger elementary school students' attitudes towards programming. *Behaviour and Information Technology*, 40(16), 1784–1795. <https://doi.org/10.1080/0144929X.2020.1781931>
- Lawshe, C. H. (1975). A quantitative approach to content validity. *Personnel Psychology*, 28(4), 563–575. <https://doi.org/10.1111/j.1744-6570.1975.tb01393.x>
- Leondari, A., & Gonida, E. (2007). Predicting academic self-handicapping in different age groups: The role of personal achievement goals and social goals. *British Journal of Educational Psychology*, 77(3), 595–611.

<https://doi.org/10.1348/000709906X128396>

Leppink, J., Paas, F., Van der Vleuten, C. P. M., Van Gog, T., & Van Merriënboer, J. J. G. (2013). Development of an instrument for measuring different types of cognitive load. *Behavior Research Methods*, 45(4), 1058–1072. <https://doi.org/10.3758/s13428-013-0334-1>

Leppink, J., & van den Heuvel, A. (2015). The evolution of cognitive load theory and its application to medical education. *Perspectives on Medical Education*, 4(3), 119–127. <https://doi.org/10.1007/s40037-015-0192-x>

Liem, A. D., Lau, S., & Nie, Y. (2008). The role of self-efficacy, task value, and achievement goals in predicting learning strategies, task disengagement, peer relationship, and achievement outcome. *Contemporary Educational Psychology*, 33(4), 486–512. <https://doi.org/10.1016/j.cedpsych.2007.08.001>

Lim, S. (2019). Implementing Social Learning for More Equitable Collaboration in Introductory Computer Science Education (Publication No. 0058F11657) [Doctoral dissertation, Cornell University]. Cornell Theses and Dissertations. <https://doi.org/10.7298/hjtz-t152>

Lishinski, A., Yadav, A., Good, J., & Enbody, R. (2016). Learning to program: Gender differences and interactive effects of students' motivation, goals, and self-efficacy on performance. *ICER 2016 - Proceedings of the 2016 ACM Conference on International Computing Education Research*, Australia, 12(1), 211–220. <https://doi.org/10.1145/2960310.2960329>

Lister, R., Fone, W., McCartney, R., Seppälä, O., Adams, E. S., Hamer, J., Moström, J. E., Simon, B., Fitzgerald, S., Lindholm, M., Sanders, K., & Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. *SIGCSE Bulletin*, 4(1), 119-150. <https://doi.org/10.1145/1041624.1041673>

Looker, N. (2021). A pedagogical framework for teaching computer programming: A social constructivist and cognitive load theory approach. *ICER 2021 - Proceedings of the 17th ACM Conference on International Computing Education Research, USA*, 17(1), 415–416. <https://doi.org/10.1145/3446871.3469778>

- Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. *ICER'08 - Proceedings of the ACM Workshop on International Computing Education Research, Australia*, 1(1), 101–111. <https://doi.org/10.1145/1404520.1404531>
- Love, T. S. (2023). Examining middle school students' attitudes toward computing after participating in a physical computing unit. *Interactive Learning Environments*, 31(1), 1–20. <https://doi.org/10.1080/10494820.2023.2194326>
- Luxton-Reilly, A. (2016). Learning to program is easy. *Annual Conference on Innovation and Technology in Computer Science Education, Peru*, 21(1), 284–289. <https://doi.org/10.1145/2899415.2899432>
- Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J., & Szabo, C. (2018). Introductory programming: A systematic literature review. *Annual Conference on Innovation and Technology in Computer Science Education, Cyprus*, 23(1), 55–106. <https://doi.org/10.1145/3293881.3295779>
- Ma, N., Qian, J., Gong, K., & Lu, Y. (2023). Promoting programming education of novice programmers in elementary schools: A contrasting cases approach for learning programming. *Education and Information Technologies*, 28(7), 9211–9234. <https://doi.org/10.1007/s10639-022-11565-9>
- Marks, G. N., Cresswell, J., & Ainley, J. (2006). Explaining socioeconomic inequalities in student achievement: The role of home and school factors. *Educational Research and Evaluation*, 12(2), 105–128. <https://doi.org/10.1080/13803610600587040>
- Mason, R., & Cooper, G. (2013). Mindstorms robots and the application of cognitive load theory in introductory programming. *Computer Science Education*, 23(4), 296–314. <https://doi.org/10.1080/08993408.2013.847152>
- Master, A., Cheryan, S., & Meltzoff, A. N. (2016). Computing whether she belongs: Stereotypes undermine girls' interest and sense of belonging in computer science. *Journal of Educational Psychology*, 108(3), 424–437. <https://doi.org/10.1037/edu0000061>
- Mathews, D. K. (2017). *Predictors of success in learning computer programming*

(Publication No. 10266241) [Doctoral dissertation, University of Rhode Island]. ProQuest Dissertations & Theses Global. <https://www.proquest.com/openview/4939e3c51db03598b9d99a0dc3e3e36f/1?pq-origsite=gscholar&cbl=18750>

Meece, J. L., Anderman, E. M., & Anderman, L. H. (2006). Classroom goal structure, student motivation, and academic achievement. *Annual Review of Psychology*, 57(1), 487–503. <https://doi.org/10.1146/annurev.psych.56.091103.070258>

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (Moti). (2013). Learning computer science concepts with Scratch. *Computer Science Education*, 23(3), 239–264. <https://doi.org/10.1080/08993408.2013.832022>

Menard, S. (2010). *Applied logistic regression analysis* (2nd ed.) Sage

Merriam, S. B. (2009). *Qualitative research: A guide to design and implementation*. John Wiley & Sons. <https://doi.org/10.1097/NCI.0b013e3181edd9b1>

Metin, S., Basaran, M., & Kalyenci, D. (2023). Examining coding skills of five-year-old children. *Pedagogical Research*, 8(2), 1-13. <https://doi.org/10.29333/pr/12802>

Midgley, C., Kaplan, A., & Middleton, M. (2001). Performance-approach goals: Good for what, for whom, under what circumstances, and at what cost? *Journal of Educational Psychology*, 93(1), 77–86. <https://doi.org/10.1037/0022-0663.93.1.77>

Midgley, C., & Urdan, T. (2001). Academic self-handicapping and achievement goals: A further examination. *Contemporary Educational Psychology*, 26(1), 61–75. <https://doi.org/10.1006/ceps.2000.1041>

Miles, M. B., & Huberman, A. M. (1994). *Qualitative Data Analysis Second Edition: Expanded Sourcebook*. Sage.

Miller, G. A. (1956). The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review*, 63(2), 81-97. <https://doi.org/10.1037/h0043158>

- Mohamad Gobil, A. R., Shukor, Z., & Mohtar, I. A. (2009). Novice difficulties in selection structure. *Proceedings of the 2009 International Conference on Electrical Engineering and Informatics, Malaysia*, 2(1), 351–356. <https://doi.org/10.1109/ICEEI.2009.5254715>
- Morrison, B. B., Dorn, B., & Guzdial, M. (2014). Measuring cognitive load in introductory CS: Adaptation of an instrument. *ICER 2014 - Proceedings of the 10th Annual International Conference on International Computing Education Research, UK*, 10(1), 131–138. <https://doi.org/10.1145/2632320.2632348>
- Muldner, K., Jennings, J., & Chiarelli, V. (2022). A Review of Worked Examples in Programming Activities. *ACM Transactions on Computing Education*, 23(1), 1–35. <https://doi.org/10.1145/3560266>
- Munzel, U., & Brunner, E. (2000). Nonparametric tests in the unbalanced multivariate one-way design. *Biometrical Journal*, 42(7), 837–854. [https://doi.org/10.1002/1521-4036\(200011\)42:7<837::AID-BIMJ837>3.0.CO;2-S](https://doi.org/10.1002/1521-4036(200011)42:7<837::AID-BIMJ837>3.0.CO;2-S)
- Myers, B. A. (1986). Visual programming, programming by example, and program visualization: a taxonomy. *ACM SIGCHI Bulletin*, 17(4), 59–66. <https://doi.org/10.1145/22339.22349>
- Nainan, M., & Balakrishnan, B. (2019). Design and Evaluation of Worked Examples for Teaching and Learning Introductory Programming at Tertiary Level. *Malaysian Online Journal of Educational Technology*, 7(4), 30–44. <https://doi.org/10.17220/mojet.2019.04.003>
- Nasution, S., Asmin, A., & Lubis, A. (2022). Analysis of mathematical problem solving ability through application of think aloud pair problem solving learning model in State Junior High School Al Manar. *Proceedings of the 7th Annual International Seminar on Transformative Education and Educational Leadership, Indonesia*, 7(1), 408-418. <https://doi.org/10.4108/eai.20-9-2022.2324713>
- Newstead, S. E., Franklyn-Stokes, A., & Armstead, P. (1996). Individual differences in student cheating. *Journal of Educational Psychology*, 88(2), 229–241. <https://doi.org/10.1037/0022-0663.88.2.229>

- Newton, S., Alemdar, M., Rutstein, D., Edwards, D., Helms, M., Hernandez, D., & Usselman, M. (2021). Utilizing evidence-centered design to develop assessments: A high school introductory computer science course. *Frontiers in Education*, 6(1), 1–19. <https://doi.org/10.3389/educ.2021.695376>
- Nicholls, J. G. (1984). Achievement motivation: Conceptions of ability, subjective experience, task choice, and performance. *Psychological Review*, 91(3), 328–346. <https://doi.org/10.1037/0033-295X.91.3.328>
- Nicholls, J. G. (1989). *The competitive ethos and democratic education*. Harvard University Press
- O'Connor, C., & Joffe, H. (2020). Intercoder Reliability in Qualitative Research: Debates and Practical Guidelines. *International Journal of Qualitative Methods*, 19(1), 1–13. <https://doi.org/10.1177/1609406919899220>
- Ober, T. M., Cheng, Y., Coggins, M. R., Brenner, P., Zdankus, J., Gonsalves, P., Johnson, E., & Urdan, T. (2024). Charting a path for growth in middle school students' attitudes toward computer programming. *Computer Science Education*, 34(1), 4–36. <https://doi.org/10.1080/08993408.2022.2134677>
- OECD. (2022). *PISA 2022 results (Vol. I-II) Country Notes: Türkiye* (OECD Publication No. d67e6c05). The Organisation for Economic Co-operation and Development (OECD). https://www.oecd.org/en/publications/pisa-2022-results-volume-i-and-ii-country-notes_ed6fbcc5-en/turkiye_d67e6c05-en.htmlhttps://
- Ouahbi, I., Kaddari, F., Darhmaoui, H., Elachqar, A., & Lahmine, S. (2015). Learning basic programming concepts by creating games with Scratch programming environment. *Procedia - Social and Behavioral Sciences*, 191(1), 1479–1482. <https://doi.org/10.1016/j.sbspro.2015.04.224>
- Özeren, E. (2022). *Proje tabanlı öğrenmede sabit ve paylaşılan liderlik uygulamaları ile desteklenmiş blok tabanlı kodlama öğretiminin bilgi işlemsel düşünmeye, güdülenmeye ve kodlama eğitime yönelik tutuma etkisi* (Publication No. 718958) [Master Thesis, Bartın University]. BU Repository. <http://hdl.handle.net/11772/16033>
- Paas, F. G. W. C., & Van Merriënboer, J. J. G. (1994). Variability of worked

- examples and transfer of geometrical problem-solving skills: A cognitive-load approach. *Journal of Educational Psychology*, 86(1), 122–133. [https://doi.org/0022-0663/94/\\$3.00](https://doi.org/0022-0663/94/$3.00)
- Paas, F., Renkl, A., & Sweller, J. (2003). Cognitive load theory and instructional design: Recent developments. *Educational Psychologist*, 38(1), 1–4. https://doi.org/10.1207/S15326985EP3801_1
- Paas, F., Renkl, A., & Sweller, J. (2004). Cognitive load theory: Instructional implications of the interaction between information structures and cognitive architecture. *Instructional Science*, 32(1–2), 1–8. <https://doi.org/10.1023/b:truc.0000021806.17516.d0>
- Paas, F., & Sweller, J. (2012). An Evolutionary Upgrade of Cognitive Load Theory: Using the Human Motor System and Collaboration to Support the Learning of Complex Cognitive Tasks. *Educational Psychology Review*, 24(1), 27–45. <https://doi.org/10.1007/s10648-011-9179-2>
- Pallant, J. (2016). *SPSS Survival Manual Survival Manual* (6th ed). McGraw-Hill Education. <https://doi.org/10.4324/9781003117452>
- Panizzon, D. (2015). Impact of Geographical Location on Student Achievement: Unpacking the Complexity of Diversity. In A. Bishop, H. Tan & T.N. Barkatsas (Eds.), *Diversity in Mathematics Education* (pp. 41–61). Springer International Publishing. https://doi.org/10.1007/978-3-319-05978-5_3
- Papatga, E., & Ersoy, A. (2016). Improving reading comprehension skills through the SCRATCH program. *International Electronic Journal of Elementary Education*, 9(1), 124–150. <https://files.eric.ed.gov/fulltext/EJ1126664.pdf>
- Perkins, D., & Martin, F. (1986). Fragile knowledge and neglected strategies in novice programmers. *Papers Presented at the First Empirical Studies of Programmers: First Workshop, USA*, 1(1), 213–229. <https://dl.acm.org/doi/abs/10.5555/21842.28896>
- Peteranetz, M. S. (2021). Shifting goals in introductory and advanced computer science courses: The effects of gender and major. *Proceedings - Frontiers in Education Conference, USA*, 51(1), 1–8. <https://doi.org/10.1109/FIE49875.2021.9637156>

- Pintrich, P. R. (2000). The Role of Goal Orientation in Self-Regulated Learning. In M. Boekaerts, P.R. Pintrich & M. Zeidner (Eds.), *Handbook of Self-Regulation*, (pp. 451–502). Academic Press. <https://doi.org/10.1016/B978-0-12-109890-2.X5027-6>
- Pokorny, K. (2009). Introduction to computing: a fresh breadth of disciplines. *Journal of Computing Sciences in Colleges*, 24(5), 166–172. <https://dl.acm.org/doi/abs/10.5555/1516595.1516630>
- Polso, K. M., Tuominen, H., Hellas, A., & Ihantola, P. (2020). Achievement Goal Orientation Profiles and Performance in a Programming MOOC. *Proceedings of the Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE, Norway*, 25(1), 411–417. <https://doi.org/10.1145/3341525.3387398>
- Presidency of the Board of Education. (n.d.). *Sıkça Sorulan Sorular*. <https://ttkb.meb.gov.tr/www/sss.php>
- Qian, Y., & Lehman, J. D. (2016). Correlates of success in introductory programming: A study with middle school students. *Journal of Education and Learning*, 5(2), 73. <https://doi.org/10.5539/jel.v5n2p73>
- Quintero-Manes, R., Vieira, C., & Hernandez-Vargas, N. (2022). Measuring cognitive loads while learning computational statistics. *Proceedings - Frontiers in Education Conference, FIE, Sweden*, 52(1), 1–4. <https://doi.org/10.1109/FIE56618.2022.9962606>
- Ramalingam, V., LaBelle, D., & Wiedenbeck, S. (2004). Self-efficacy and mental models in learning to program. *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, USA*, 9(1), 171–175. <https://doi.org/10.1145/1007996.1008042>
- Ramalingam, V., & Wiedenbeck, S. (1998). Development and validation of scores on a computer programming self-efficacy scale and group analyses of novice programmer self-efficacy. *Journal of Educational Computing Research*, 19(4). <https://doi.org/10.2190/C670-Y3C8-LTJ1-CT3P>
- Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009).

- Scratch: Programming for all. *Communications of the ACM*, 52(11), 60–67.
<https://doi.org/10.1145/1592761.1592779>
- Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *International Journal of Phytoremediation*, 21(1), 137–172.
<https://doi.org/10.1076/csed.13.2.137.14200>
- Rumelhart, D. E., & Norman, D. A. (1976). Accretion, Tuning and Restructuring: three modes of learning. (Nº 7602). *Personnel and Training Research Programs Office of Naval Research*, 7602(1), 37–53.
<https://files.eric.ed.gov/fulltext/ED134902.pdf>
- Salac, J., Thomas, C., Butler, C., & Franklin, D. (2021). Understanding the Link between Computer Science Instruction and Reading and Math Performance. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE, Germany*, 26(1), 408–414.
<https://doi.org/10.1145/3430665.3456313>
- Saldaña, J. (2009). *Qualitative Researchers*.
http://stevescollection.weebly.com/uploads/1/3/8/6/13866629/saldana_2009_the-coding-manual-for-qualitative-researchers.pdf
- Salleh Hudin, S. (2023). A Systematic Review of the Challenges in Teaching Programming for Primary Schools’ Students. *Online Journal for TVET Practitioners*, 8(1), 75–88. <https://doi.org/10.30880/ojtp.2023.08.01.008>
- Sands, P. (2019). Addressing cognitive load in the computer science classroom. *ACM Inroads*, 10(1), 44–51. <https://doi.org/10.1145/3210577>
- Schoeman, M. (2019). Reading skills can predict the-programming performance of novices: An eye-Tracking study. *Perspectives in Education*, 37(2), 35–52.
<https://doi.org/10.18820/2519593X/pie.v37i2.3>
- Schulte, C., & Bennedsen, J. (2006). What do teachers teach in introductory programming? *ICER 2006 - Proceedings of the 2nd International Computing Education Research Workshop, USA2*(1), 17–28.
<https://doi.org/10.1145/1151588.1151593>

- Schulz, S., Berndt, S., & Hawlitschek, A. (2023). Exploring students' and lecturers' views on collaboration and cooperation in computer science courses - a qualitative analysis. *Computer Science Education*, 33(3), 318–341. <https://doi.org/10.1080/08993408.2021.2022361>
- Schunk, D. H., & DiBenedetto, M. K. (2020). Motivation and social cognitive theory. *Contemporary Educational Psychology*, 60(1), 1–10. <https://doi.org/10.1016/j.cedpsych.2019.101832>
- Schwinger, M., Wirthwein, L., Lemmer, G., & Steinmayr, R. (2014). Academic self-handicapping and achievement: A meta-analysis. *Journal of Educational Psychology*, 106(3), 744–761. <https://doi.org/10.1037/a0035832>
- Senko, C., Hulleman, C. S., & Harackiewicz, J. M. (2011). Achievement goal theory at the crossroads: Old controversies, current challenges, and new directions. *Educational Psychologist*, 46(1), 26–47. <https://doi.org/10.1080/00461520.2011.538646>
- Shell, D. F., Hazley, M. P., Soh, L.-K., Ingraham, E., & Ramsay, S. (2013). Associations of students' creativity, motivation, and self-regulation with learning and achievement in college computer science courses. *2013 IEEE Frontiers in Education Conference (FIE), USA*, 43(1), 1637–1643. <https://doi.org/10.1109/FIE.2013.6685116>
- Shell, D. F., & Soh, L. K. (2013). Profiles of Motivated Self-Regulation in College Computer Science Courses: Differences in Major versus Required Non-Major Courses. *Journal of Science Education and Technology*, 22(6), 899–913. <https://doi.org/10.1007/s10956-013-9437-9>
- Shell, D. F., Soh, L. K., Flanigan, A. E., & Peteranetz, M. S. (2016). Students' initial course motivation and their achievement and retention in college cs1 courses. *SIGCSE 2016 - Proceedings of the 47th ACM Technical Symposium on Computing Science Education, USA*, 47(1), 639–644. <https://doi.org/10.1145/2839509.2844606>
- Skaalvik, E. M. (1997). Self-enhancing and self-defeating ego orientation: Relations with task and avoidance orientation, achievement, self-perceptions, and anxiety. *Journal of Educational Psychology*, 89(1), 71–81. <https://doi.org/10.1037//0022-0663.89.1.71>

- Sleeman, D., Putnam, R. T., Baxter, J., & Kuspa, L. (1984). Pascal and high-school students: A study of misconceptions. *Journal of Educational Computing Research*, 2(1), 5–23. http://server4.isearch-it-solutions.net:80/pubpsych/Search.action?q=ID%3DACCNO_ED258552&isFullView=true&stats=BMD&search=
- Soltani, A., Boka, R. S., & Jafarzadeh, A. (2022). Students' perceptions of learning environment: associations with personal mastery goal orientations, regulations, and academic performance in biology. *International Journal of Science Education*, 44(9), 1462–1480. <https://doi.org/10.1080/09500693.2022.2082578>
- Sullivan, A., & Bers, M. U. (2016). Girls, boys, and bots: Gender differences in young children's performance on robotics and programming tasks. *Journal of Information Technology Education: Innovations in Practice*, 15(1), 145–165. <https://doi.org/10.28945/3547>
- Sun, L., Hu, L., & Zhou, D. (2022). Programming attitudes predict computational thinking: Analysis of differences in gender and programming experience. *Computers and Education*, 181(27), 104457. <https://doi.org/10.1016/j.compedu.2022.104457>
- Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science*, 12(2), 257–285. [https://doi.org/10.1016/0364-0213\(88\)90023-7](https://doi.org/10.1016/0364-0213(88)90023-7)
- Sweller, J. (2010). Element interactivity and intrinsic, extraneous, and germane cognitive load. *Educational Psychology Review*, 22(2), 123–138. <https://doi.org/10.1007/s10648-010-9128-5>
- Sweller, J. (2016). Working memory, long-term memory, and instructional design. *Journal of Applied Research in Memory and Cognition*, 5(4), 360–367. <https://doi.org/10.1016/j.jarmac.2015.12.002>
- Sweller, J. (2020). Cognitive load theory and educational technology. *Educational Technology Research and Development*, 68(1), 1–16. <https://doi.org/10.1007/s11423-019-09701-3>
- Sweller, J., Ayres, P., & Kalyuga, S. (2011). *Cognitive load theory*. Springer.

<https://doi.org/10.1007/978-1-4419-8126-4>

- Sweller, J., & Cooper, G. A. (1985). The use of worked examples as a substitute for problem solving in learning algebra. *Cognition and Instruction*, 2(1), 59–89. https://doi.org/10.1207/s1532690xc0201_3
- Sweller, J., Merrienboer, J. J. G. van, & Paas, F. G. W. C. (1998). Cognitive architecture and instructional design. *Educational Psychology Review*, 10(3), 251–296. https://noic.com.br/wp-content/uploads/2022/02/Worked_Examples.pdf
- Sweller, J., van Merriënboer, J. J. G., & Paas, F. (2019). Cognitive architecture and instructional design: 20 years later. *Educational Psychology Review*, 31(2), 261–292. <https://doi.org/10.1007/s10648-019-09465-5>
- Tabachnick, B. G., & Fidell, L. S. (2012). *Using multivariate statistics* (6th ed.). Harper and Row.
- Takir, A. (2011). *The effect of an instruction designed by cognitive load theory principles on 7th grade students' achievement in algebra topics and cognitive load* (Publication No. 300705) [Doctoral Dissertation, Middle East Technical University]. ODTU Repository. <https://go.exlibris.link/KszTQDfd>
- Taşdöndüren, T., & Korucu, A. T. (2022). The effect of secondary school students' perceptions of computing technologies and self-efficiency on attitudes towards coding. *Journal of Learning and Teaching in Digital Age*, 7(2), 200–209. <https://doi.org/10.53850/joltida.1035448>
- 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*, 12(2), 1-29. <https://doi.org/10.1145/2160547.2160551>
- Tellhed, U., Björklund, F., & Kallio Strand, K. (2022). Sure I can code (but do I want to?). Why boys' and girls' programming beliefs differ and the effects of mandatory programming education. *Computers in Human Behavior*, 135(1), 1-11. <https://doi.org/10.1016/j.chb.2022.107370>
- Thomas, M. K., & Greene, B. A. (2011). Fostering 21st century skill development by engaging students in authentic game design projects in a high school

- computer programming class. *Journal of Educational Computing Research*, 44(4), 383–400. <https://doi.org/10.2190/EC.44.4.b>
- Toma, L., & Vahrenhold, J. (2018). Self-efficacy, cognitive load, and emotional reactions in collaborative algorithms labs - A case study. *ICER 2018 - Proceedings of the 2018 ACM Conference on International Computing Education Research, USA*, 14(1), 1–10. <https://doi.org/10.1145/3230977.3230980>
- Tomić, B., Milikić, N., Jovanović, J., & Devedžić, V. (2020). Examining attendance , performance and interest in a CS course in relation to students' achievement goal orientation and self- evaluation. *International Conference on Information Technology and Development of Education, Serbia*, 11(1), 1.7. <http://www.tfzr.rs/itro/arhiva/itro/FILES/21.PDF>
- Totan, H. N., & Korucu, A. T. (2023). The effect of block based coding education on the students' attitudes about the secondary school students' computational learning skills and coding learning: Blocky sample. *Participatory Educational Research*, 10(1), 443–461. <https://doi.org/10.17275/per.23.24.10.1>
- 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/10.1177/0735633117746747>
- Turner, J. C., Midgley, C., Meyer, D. K., Gheen, M., Anderman, E. M., Kang, Y., & Patrick, H. (2002). The classroom environment and students' reports of avoidance strategies in mathematics: A multimethod study. *Journal of Educational Psychology*, 94(1), 88–106. <https://doi.org/10.1037/0022-0663.94.1.88>
- Urduan, T. (2004). Predictors of academic self-handicapping and achievement: Examining achievement goals, classroom goal structures, and culture. *Journal of Educational Psychology*, 96(2), 251–264. <https://doi.org/10.1037/0022-0663.96.2.251>
- Urduan, T., & Midgley, C. (2001). Academic self-handicapping: What we know, what more there is to learn. *Educational Psychology Review*, 13(2), 115–138. <https://doi.org/10.1023/A:1009061303214>

- Urdan, T., & Midgley, C. (2003). Changes in the perceived classroom goal structure and pattern of adaptive learning during early adolescence. *Contemporary Educational Psychology*, 28(4), 524–551. [https://doi.org/10.1016/S0361-476X\(02\)00060-7](https://doi.org/10.1016/S0361-476X(02)00060-7)
- Urdan, T., Midgley, C., & Anderman, E. M. (1998). The role of classroom goal structure in students' use of self-handicapping strategies. *American Educational Research Journal*, 35(1), 101–122. <https://doi.org/10.3102/00028312035001101>
- Usher, E. L. (2009). Sources of middle school students' self-efficacy in mathematics: A qualitative investigation. *American Educational Research Journal*, 46(1), 275–314. <https://doi.org/10.3102/0002831208324517>
- Van Merriënboer, J. J. G., & Krammer, H. P. M. (1987). Instructional strategies and tactics for the design of introductory computer programming courses in high school. *Instructional Science*, 16(3), 251–285. <https://doi.org/10.1007/BF00120253>
- van Merriënboer, J. J. G., & Sweller, J. (2005). Cognitive Load theory and complex learning: Recent developments and future directions. *Educational Psychology Review*, 17(2), 147–177. <https://doi.org/10.1007/s10648-005-3951-0>
- Van Merriënboer, J. J. G., & Sweller, J. (2010). Cognitive load theory in health professional education: Design principles and strategies. *Medical Education*, 44(1), 85–93. <https://doi.org/10.1111/j.1365-2923.2009.03498.x>
- Vandenberg, J., Rachmatullah, A., Lynch, C., Boyer, K. E., & Wiebe, E. (2021). The relationship of cs attitudes, perceptions of collaboration, and pair programming strategies on upper elementary students' CS learning. *Proceedings of the Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE, USA, 26(1), 46–52. <https://doi.org/10.1145/3430665.3456347>
- Wang, J., Hong, H., Ravitz, J., & Ivory, M. (2015). Gender differences in factors influencing pursuit of computer science and related fields. *Proceedings of the Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE, USA, 20(1), 117–122. <https://doi.org/10.1145/2729094.2742611>

- Watson, C., & Li, F. W. B. (2014). Failure rates in introductory programming revisited. *ITICSE 2014 - Proceedings of the 2014 Innovation and Technology in Computer Science Education Conference, USA, 19(1)*, 39–44. <https://doi.org/10.1145/2591708.2591749>
- Webb, M., Davis, N., Bell, T., Katz, Y., Reynolds, N., Chambers, D. P., & Sysło, M. M. (2017). Computer science in K-12 school curricula of the 21st century: Why, what and when? *Education and Information Technologies, 22(2)*, 445–468. <https://doi.org/10.1007/s10639-016-9493-x>
- Weintrop, D., & Wilensky, U. (2015). To block or not to block , that is the question : Students’ perceptions of blocks-based programming. *Proceedings of the 14th International Conference on Interaction Design and Children, Denmark, 14(1)*, 199–208.
- Wells LeRoy, A. K. (2022). *A mixed methods approach to understanding the effect of applying multimedia principles to a Minecraft STEM lesson* (Publication No. 29395202) [Doctoral Dissertation, University of California]. ProQuest Dissertations & Theses Global.
- Weng, C., Otanga, S., Weng, A., & Cox, J. (2018). Effects of interactivity in E-textbooks on 7th graders science learning and cognitive load. *Computers and Education, 120(1)*, 172–184. <https://doi.org/10.1016/j.compedu.2018.02.008>
- White, G., & Ploeger, F. (2004). Cognitive characteristics for learning visual basic. *Journal of Computer Information Systems, 44(3)*, 58–66. <https://doi.org/10.1080/08874417.2004.11647582>
- Williams, L. (1999). But, Isn’t That Cheating ? *Proceedings of the 29th Annual Frontiers in Education Conference, USA, 29(1)*, 26–27. <https://www.computer.org/csdl/proceedings/fie/1999/12OmNC8dg8Z>
- Williams, L., & Upchurch, R. L. (2001). In support of student pair-programming. *SIGCSE Bulletin (Association for Computing Machinery, Special Interest Group on Computer Science Education), USA, 52(1)*, 327–331. <https://doi.org/10.1145/366413.364614>
- Winslow, L. E. (1996). Programming pedagogy - A psychological overview. *SIGCSE Bulletin (Association for Computing Machinery, Special Interest*

Group on Computer Science Education), 28(3), 17–23.
<https://doi.org/10.1145/234867.234872>

Winter, V., Friend, M., Matthews, M., Love, B., & Vasireddy, S. (2019). Using visualization to reduce the cognitive load of threshold concepts in computer programming. *Proceedings - Frontiers in Education Conference, FIE, USA*, 49(1), 1–9. <https://doi.org/10.1109/FIE43999.2019.9028612>

Yang, K. H., & Lin, H. Y. (2019). Exploring the effectiveness of learning Scratch programming with Code.org. *Proceedings of the 8th International Congress on Advanced Applied Informatics, Japan*, 8(1), 1057–1058.
<https://doi.org/10.1109/IIAI-AAI.2019.00225>

Yeşilyurt, S., & Çapraz, C. (2018). *A road map for the content validity used in scale development studies. Erzincan Üniversitesi Eğitim Fakültesi Dergisi*, 20(1), 251–264. <https://doi.org/10.17556/erziefd.297741>

Yusof, M. M., Jalil, H. A., & Perumal, T. (2021). Exploring teachers' practices in teaching robotics programming in primary school. *Asian Social Science*, 17(11), 122. <https://doi.org/10.5539/ass.v17n11p122>

Zingaro, D. (2014). Peer instruction contributes to self-efficacy in CS1. *Proceedings of the 45th ACM Technical Symposium on Computer Science Education, USA*, 45(1), 373–378. <https://doi.org/10.1145/2538862.2538878>

Zingaro, D., Craig, M., Porter, L., Becker, B. A., Cao, Y., Conrad, P., Cukierman, D., Hellas, A., Loksa, D., & Thota, N. (2018). Achievement goals in CS1: Replication and extension. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education, USA*, 49(1), 687–692.
<https://doi.org/10.1145/3159450.3159452>



APPENDICES

A. Lesson Plan Evaluation Form

Bölüm 1: Demografik Bilgiler

Adınız-Soyadınız	
Cinsiyetiniz	☐ Kadın ☐ Erkek
Yaşınız	
Mesleki deneyiminiz (yıl olarak)	
Eğitim düzeyiniz	☐ Lisans ☐ Yüksek Lisans ☐ Doktora ☐ Diğer:
Mezun olduğunuz lisans programı	☐ Bilgisayar ve Öğretim Teknolojileri Öğretmenliği ☐ Bilgisayar Öğretmenliği ☐ Bilgisayar Sistemleri Öğretmenliği ☐ Bilgisayar ve Kontrol Öğretmenliği ☐ Elektronik ve Bilgisayar Öğretmenliği ☐ Bilgisayar Teknolojisi Bölümü ☐ Bilgisayar Teknolojisi ve Bilişim Sistemleri Bölümü
Görev yapmakta olduğunuz okul türü	☐ Devlet ☐ Özel
Görev yapmakta olduğunuz okulda bilgisayar laboratuvarı mevcut mu?	☐ Evet ☐ Hayır
Ortaokul öğrencilerine programlama öğretimi deneyiminiz (yıl olarak)	

Bölüm 2- Ders Planı Değerlendirme

Ders Planı:	Evet/Hayır	Açıklama
Bölüm1		
code.org sitesinde yer alan bu ders planını daha önce derslerinde kullanmış mıydınız?	☐ Evet ☐ Hayır	
Kazanımlar açık ve anlaşılır bir şekilde belirtilmiş mi?	☐ Evet ☐ Hayır	
Kazanımlar öğrenci düzeyine uygun mu?	☐ Evet ☐ Hayır	
Kazanımlar belirlenen sürede ulaşılabilir mi?	☐ Evet ☐ Hayır	

Kazanımlar Ortaokul Bilişim Teknolojileri ve Yazılım Dersi Öğretim Programının amaçları ile uyuyor mu?	☐ Evet ☐ Hayır	
Planda yer alan anahtar kelimeler doğru ve anlaşılır bir şekilde tanımlanmış mı?	☐ Evet ☐ Hayır	
Ders planı, planda belirtilen sürede tamamlandı mı?	☐ Evet ☐ Hayır	
İçerik, kazanımlar ile uyumlu bir şekilde planlanmış mı?	☐ Evet ☐ Hayır	
Derse hazırlık sürecinde zorlandınız mı?	☐ Evet ☐ Hayır	
Ders planı, öğrenci seviyesine ve konuya uygun araç – gereçleri içeriyor mu?		
Etkinlikler öğrenci düzeylerine uygun mu?	☐ Evet ☐ Hayır	
Etkinlikler konuya uygun mu?	☐ Evet ☐ Hayır	
Etkinliklerin ne zaman ve nasıl gerçekleştirileceği açık ve anlaşılır mı?	☐ Evet ☐ Hayır	
Etkinlikler öğrencilerin dikkatini çekme konusunda etkili mi?	☐ Evet ☐ Hayır	
Etkinlikler öğrencilerin dikkatinin devamını sağlama konusunda etkili mi?	☐ Evet ☐ Hayır	
Bölüm 2		
Ders planından çıkartmak istediğiniz herhangi bir şey var mı? Varsa nedenleriyle birlikte belirtiniz.		
Ders planında yeniden düzenlemek istediğiniz bir bölüm var mı? Varsa neden ve ne şekilde bir düzenleme yapmak istersiniz?		
Ders planına eklemek istediğiniz herhangi bir şey var mı?		
Ders planında uygulamasını zor bulduğunuz bir bölüm oldu mu? Açıklayınız.		
Öğrencilerin anlamakta/uygulamakta zorluk çektiklerini düşündüğünüz bir bölüm oldu mu? Açıklayınız.		
Ders planı ile ilgili genel bir değerlendirmede bulunur musunuz?		
Eklemek istediğiniz başka bir şey var mı?		

B. Coding Achievement Test

AD-SOYAD:

1

Bilişim Teknolojileri ve Yazılım Dersi 5. Sınıf Kodlama Başarı Testi

Sınav Yönergesi: Bu test, kodlama konusundaki bilgi düzeyinizi ölçmek amacıyla hazırlanmıştır. Testte 36 adet çoktan seçmeli soru bulunmaktadır. Her sorunun yalnız bir doğru cevabı bulunmaktadır. Soruların cevaplarını, sınav kağıdının en sonundaki Cevap Alanına "X" ile işaretleyiniz. Katılımınız için teşekkür ederiz.

Öğr. Gör. Pınar KEFELİ BERBER
İletişim: pinar.kefeli@erdogan.edu.tr

1. Kodlamada, verileri saklamak amacıyla kullanılan ve değeri değiştirilebilen yapılar aşağıdakilerden hangisidir?

- a) Problem b) Sabit c) Algoritma d) Değişken

2. Aşağıdakilerden hangisi sabit yapısına örnek olarak verilemez?

- a) Futboldaki oyuncu sayısı
b) Trafikteki araç sayısı
c) Bir eldeki parmak sayısı
d) Bir yıldaki ay sayısı

3. Aşağıdakilerden hangisi matematiksel (aritmetiksel) operatörlere örnek olarak verilemez?

- a) + b) - c) ! d) /

4. Aşağıdakilerden hangisi mantıksal operatörlere örnek olarak verilemez?

- a) veya b) ve c) değil d) eğer



5. Karar yapıları, içerisinde verilen koşul sağlandığında "Doğru" değerini verir. Buna göre yanda verilen kod bloğu için aşağıdaki ifadelerden hangisi doğrudur?

- a) puan değişkeninin değeri 49 ise "Doğru" değerini verir.
b) puan değişkeninin değeri 50 ise "Doğru" değerini verir.
c) puan değişkeninin değeri 51 ise "Doğru" değerini verir.
d) puan değişkeninin değeri 50 ya da 51 ise "Doğru" değerini verir.

6. "Bir problemin çözümünde izlenmesi gereken adımların, sıralı bir şekilde ifade edilmesidir." ifadesi aşağıdaki kavramlardan hangisini açıklamaktadır?

- a) Algoritma b) Program c) Veri d) Operatör

7. Aşağıda verilen adımlarından hangisinin bir algoritmada mutlaka bulunması gerekmektedir?

- a) Başla b) X kez tekrarla
c) Eğer büyükse d) Sonucu ekrana yazdır.

8. Bilgisayarda kod yazan ve komutları bilgisayara ileten kişi aşağıdakilerden hangisidir?

- a) Kod b) Komut c) Programcı d) Algoritma

9. "Bir cihazın ya da yazılımın gerçekleştirilmesi istenilen işlemlere ilişkin yönergelerin, bilgisayara, programlama dili komutları halinde girilmesidir." ifadesi aşağıdakilerin hangisini tanımlamaktadır?

- a) Programlama (kodlama) b) Akış
c) Bilgisayar programcısı d) Koşul

10. Bir bilgisayar oyununda, "puan > 500 ve hamle_sayısı < 20" koşulunu sağlayan oyuncu bir üst seviyeye geçmektedir. Buna göre aşağıda bilgileri verilen oyunculardan hangisi bir üst seviyeye geçer?

Oyuncu Bilgileri		
oyuncunun adı	puan	hamle sayısı
Dilek	500	20
Ahmet	600	25
Leyla	700	15
Sena	400	5

- a) Dilek
b) Ahmet
c) Sena
d) Leyla

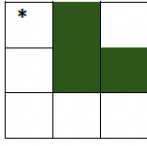
11. Aşağıda, akış şeması oluştururken kullanılan semboller ve yanlarında görevleri verilmiştir. Buna göre hangi eşleştirme yanlıştır?

- a)  Mantıksal karşılaştırma (koşullu karşılaştırma) ifadelerini göstermek için kullanılır.
b)  Başla ve bitir komutlarını temsil eder.
c)  Tekrar eden komutları belirtir.
d)  Dışarıdan bilgi/veri girişini belirler.

12. 10 kez tekrar etmesini istediğimiz olaylar için aşağıdaki kod bloklarından hangisini kullanmamız uygun olur?

- a) 
b) 
c) 
d) 

Arka sayfaya geçiniz.



13. Yandaki şekilde başlangıç konumu, * sembolü ile belirtilen karedir. Bu noktadan başlayarak, aşağıdaki algoritmaların hangisi ile yanda verilen çizim elde edilebilir?

- | | |
|---|--|
| a) Adım 1: Başla.
Adım 2: Bir kare sağa git.
Adım 3: Kareyi boy.
Adım 4: Bir kare aşağı git.
Adım 5: Kareyi boy.
Adım 6: Bir kare sağa git.
Adım 7: Kareyi boy.
Adım 8: Bitir. | b) Adım 1: Başla.
Adım 2: Bir kare sağa git.
Adım 3: Kareyi boy.
Adım 4: Bir kare aşağı git.
Adım 5: Kareyi boy.
Adım 6: Bir kare sola git.
Adım 7: Kareyi boy.
Adım 8: Bitir. |
| c) Adım 1: Başla.
Adım 2: Bir kare sola git.
Adım 3: Kareyi boy.
Adım 4: Bir kare aşağı git.
Adım 5: Kareyi boy.
Adım 6: Bir kare sağa git.
Adım 7: Kareyi boy.
Adım 8: Bitir. | d) Adım 1: Başla.
Adım 2: Bir kare sağa git.
Adım 3: Kareyi boy.
Adım 4: Bir kare aşağı git.
Adım 5: Kareyi boy.
Adım 6: Bir kare aşağı git.
Adım 7: Kareyi boy.
Adım 8: Bitir. |

- I. Ayrarı bardağa doldur.
II. Yoğurdun üzerine su ekle.
III. Bitir.
IV. Yoğurdu kaba koy.
V. Başla.
VI. Yoğurt ve suyu çürp.

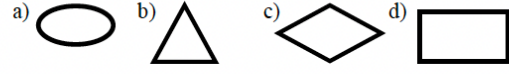
14. Yukarıda karışık sırada verilen ayran yapma algoritmasının doğru sıralanışı aşağıdakilerin hangisinde verilmiştir?

- a) V-VI-IV-II-III-I b) V-IV-II-VI-I-III
c) V-IV-II-VI-III-I d) V-IV-II-I-VI-III

15. Aşağıdakilerden hangisi girilen iki sayının ortalamasını hesaplayıp ekrana yazdıran algoritmadır?

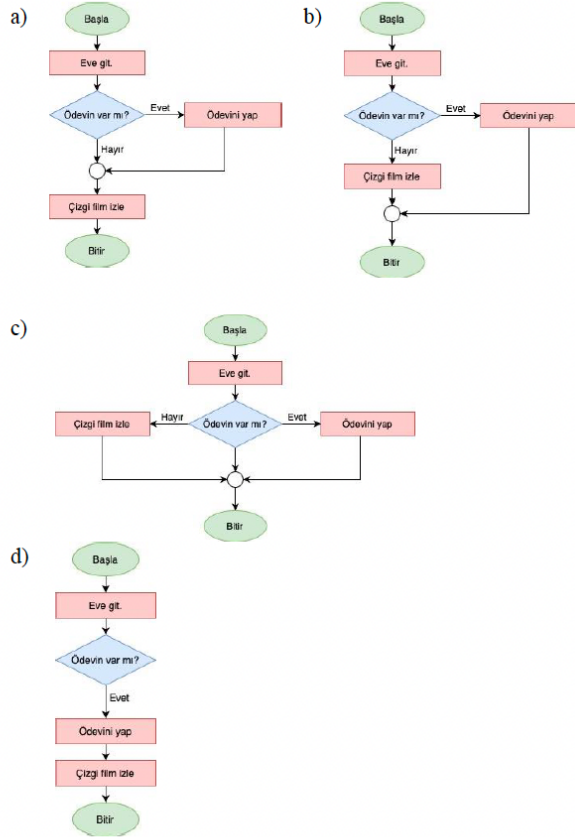
- | | |
|--|--|
| a) Adım 1: Başla.
Adım 2: Birinci sayıyı gir.
Adım 3: İkinci sayıyı gir.
Adım 4: İki sayıyı topla.
Adım 5: Toplamı ikiye böl.
Adım 6: Bitir. | b) Adım 1: Başla.
Adım 2: Birinci sayıyı gir.
Adım 3: İkinci sayıyı gir.
Adım 4: İki sayıyı topla.
Adım 5: Toplamı ikiye böl.
Adım 6: Sonucu ekrana yazdır. |
| c) Adım 1: Başla.
Adım 2: Birinci sayıyı gir.
Adım 3: İkinci sayıyı gir.
Adım 4: İki sayıyı topla.
Adım 5: Toplamı ikiye böl.
Adım 6: Sonucu ekrana yazdır.
Adım 7: Bitir. | d) Adım 1: Başla.
Adım 2: Birinci sayıyı gir.
Adım 3: İkinci sayıyı gir.
Adım 4: İki sayıyı topla.
Adım 5: Sonucu ekrana yazdır.
Adım 6: Bitir. |

16. Aşağıdaki şekillerden hangisi akış şeması oluştururken kullanılan sembollerden biri değildir?



Adım 1: Başla
Adım 2: Eve git
Adım 3: EĞER ödevin varsa 4. adıma git, YOKSA 5. adıma git.
Adım 4: Ödevini yap.
Adım 5: Çizgi film izle.
Adım 6: Bitir

17. Yanda verilen algoritmaya en uygun akış şeması aşağıdakilerden hangisidir?



18. Çalıştığı zaman Yanda verilen blok altında yer alan bloklar hangi olay gerçekleştiğinde çalışır?

- a) Sürekli olarak
b) Adım düğmesine tıklandığında
c) Varsayılan sıfırla düğmesine tıklandığında
d) Çalıştır düğmesine tıklandığında

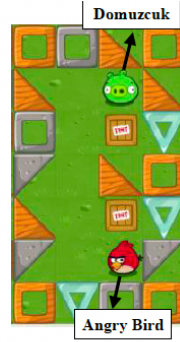
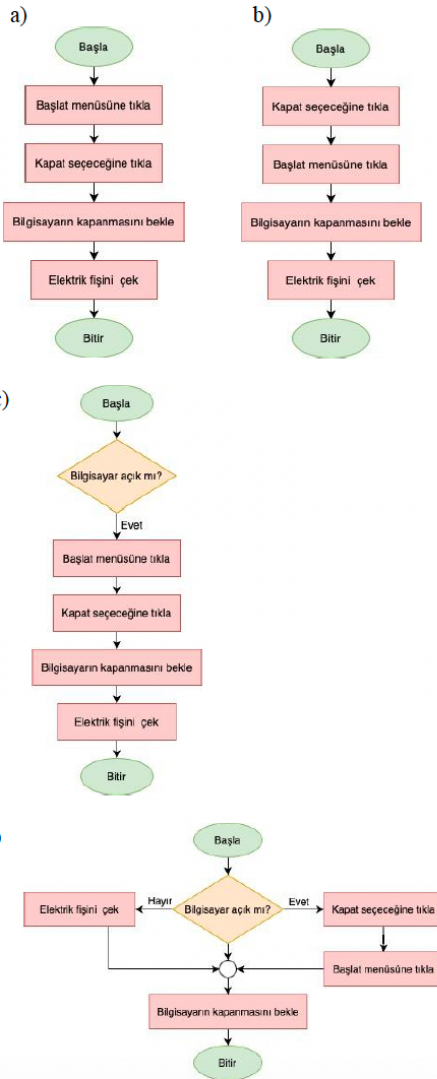
Diğer sayfaya geçiniz.

19. Kodlamada (bilgisayar programlamada), bir şeyi tekrar tekrar yapma, bir kodu tekrarlama eylemi aşağıdakilerden hangisi ile ifade edilir?

a) Kod b) Döngü c) Program d) Hata

Adım 1: Başla.
Adım 2: Başlat menüsüne tıkla.
Adım 3: Kapat seçeneğini tıkla.
Adım 4: Bilgisayarın kapanmasını bekle.
Adım 5: Elektrik fişini çek.
Adım 5: Bitir.

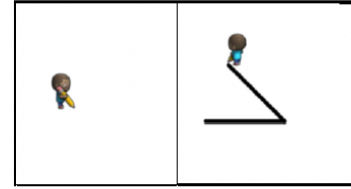
20. Yukarıda verilen algoritmaya en uygun akış şeması aşağıdakilerden hangisidir?



Adım 1: Başla
Adım 2: İlerle
Adım 3:
Adım 4: İlerle
Adım 5: İlerle
Adım 6: İlerle
Adım 7: İlerle
Adım 8: Sağa dön
Adım 9: İlerle

21. Yukarıda Angry Bird karakterinin, domuzcuğa ulaşabilmesi için gerekli olan algoritma verilmiştir. Buna göre algoritmada “.....” ile belirtilen “Adım 3” yerine aşağıdakilerden hangisi gelmelidir?

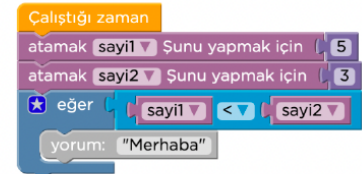
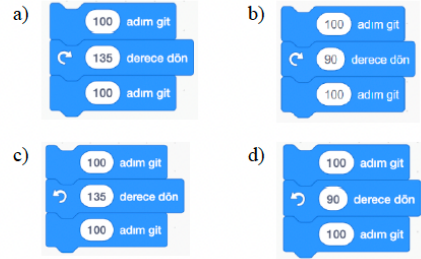
a) İlerle
b) Sağa dön
c) Sola dön
d) 3. adıma gerek yok, algoritma doğrudur.



Şekil A

Şekil B

22. Aşağıdaki kod bloklarından hangisi, başlangıç yönü Şekil-A'daki görselde verilen Artist karakterine, Şekil-B'de verilen görseldeki gibi 45 derecelik bir açı çizdirir?



23. Yukarıda verilen kod bloğu setine göre, programın “Merhaba” yorumu yapması için [sayi2]’nin değeri, aşağıdakilerden hangisi ile değiştirilmelidir?

a) 2 b) 4 c) 5 d) 6

Arka sayfaya geçiniz.

```

Çalıştığı zaman
ilerle
ilerle
sağa dön 90
ilerle

```

24. Yanda verilen kod bloğu setinde; izlenecek adımların sırayla, birbiri ardına gerçekleştirilecek şekilde tanımlandığı görülmektedir. Buna göre, bu kod bloğu setinde kullanılan yapı aşağıdakilerden hangisidir?

- a) Karar yapısı
- b) Döngü yapısı
- c) Doğrusal mantık yapısı
- d) Doğrusal olmayan mantık yapısı

25. Aşağıdaki kod bloğu seti, “nektar = 4” için çalıştırılırsa sonuç ne olur?

```

Çalıştığı zaman
eğer nektar > 5
yap nektarı al
değilse ilerle

```

- a) Nektarı alır.
- b) İlerler.
- c) 5 kez ilerler.
- d) Önce nektarı alır, sonra ilerler.



26. Yanda verilen görsel göre çiçekte nektar olup olmadığı bilinmemektedir. Arının, çiçekte bir nektar var ise nektarı alacak yoksa ilerleyecek şekilde hareket etmesini sağlayan kod bloğu aşağıdakilerden hangisidir?

```

Çalıştığı zaman
ilerle
eğer nektar = 1
yap nektarı al
değilse ilerle

```

a)

```

Çalıştığı zaman
ilerle
eğer nektar = 1
yap ilerle
değilse nektarı al

```

b)

```

Çalıştığı zaman
ilerle
eğer nektar = 1
yap nektarı al
ilerle

```

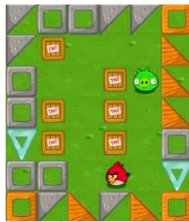
c)

```

Çalıştığı zaman
ilerle
eğer nektar = 1
yap ilerle
nektarı al

```

d)



```

Çalıştığı zaman
?
sola dön 90
ilerle
sola dön 90
ilerle
ilerle
ilerle

```

27. Yukarıdaki görselde verilen kırmızı kuşun (Angry Bird karakterinin), yeşil domuzcuğa en kısa yoldan ulaşabilmesi için soru işareti (?) ile belirtilen boşluğa hangi kod bloğu gelmelidir?

- a) Sağa dön
- b) Sola dön
- c) İlerle
- d) Hiçbir şey.

```

Çalıştığı zaman
ileriyeye taşı 50 pikseller
kadar sağa dön 90 derece
ileriyeye taşı 50 pikseller
kadar sola dön 90 derece
ileriyeye taşı 50 pikseller
kadar sağa dön 90 derece
ileriyeye taşı 50 pikseller
kadar sola dön 90 derece

```

28. Yanda, iki basamaklı merdiven çizdiren kod bloğu seti verilmiştir. Bu kod bloğu seti, döngü yapısı kullanılarak yeniden düzenlenmek istenmektedir. Buna göre en doğru çözüm aşağıdakilerden hangisidir?

```

Çalıştığı zaman
bu işlemleri 2 kez tekrarla
yap ileriyeye taşı 50 pikseller
kadar sağa dön 90 derece

```

a)

```

Çalıştığı zaman
bu işlemleri 2 kez tekrarla
yap ileriyeye taşı 50 pikseller
kadar sola dön 90 derece

```

b)

```

Çalıştığı zaman
bu işlemleri 2 kez tekrarla
yap ileriyeye taşı 50 pikseller
kadar sağa dön 90 derece
ileriyeye taşı 50 pikseller
kadar sola dön 90 derece

```

c)

```

Çalıştığı zaman
bu işlemleri 4 kez tekrarla
yap ileriyeye taşı 50 pikseller
kadar sağa dön 90 derece
ileriyeye taşı 50 pikseller
kadar sola dön 90 derece

```

d)



29. Yanda verilen görseldeki Zombi karakterinin, ayçiçeğine ulaşmasını sağlayan blok seti aşağıdakilerden hangisidir?

```

Çalıştığı zaman
bu işlemleri 6 kez tekrarla
yap sola dön 90
ilerle
sağa dön 90
ilerle

```

a)

```

Çalıştığı zaman
bu işlemleri 3 kez tekrarla
yap sola dön 90
ilerle
sağa dön 90
ilerle

```

b)

```

Çalıştığı zaman
bu işlemleri 3 kez tekrarla
yap sola dön 90
ilerle
sağa dön 90
ilerle

```

c)

```

Çalıştığı zaman
sola dön 90
ilerle
bu işlemleri 3 kez tekrarla
yap sağa dön 90
ilerle

```

d)

```

Çalıştığı zaman
bu işlemleri 3 kez tekrarla
yap bu işlemleri 4 kez tekrarla
yap ilerle
sağa dön 90

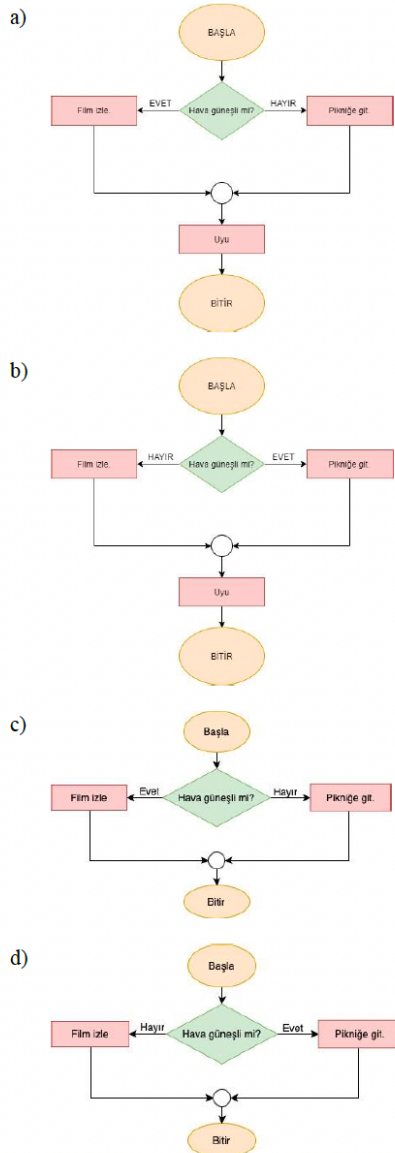
```

30. Yukarıda verilen kod bloğu setine göre ilerle kod bloğu kaç kez çalışır?

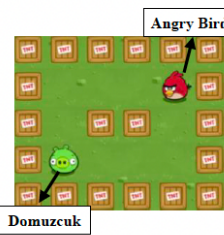
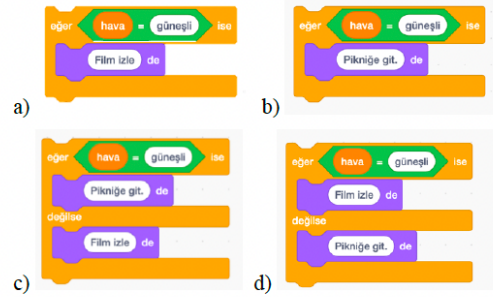
- a) 3
- b) 4
- c) 7
- d) 12

Diğer sayfaya geçiniz.

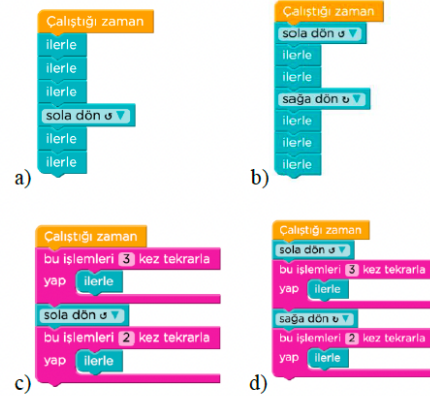
31. Bu duruma en uygun şekilde hazırlanmış akış şeması aşağıdakilerden hangisidir?



32. Bu duruma en uygun şekilde hazırlanmış kod bloğu aşağıdakilerden hangisidir?



33. Aşağıda verilen kod bloğu setlerinden hangisi, yanda görülen Angry Bird karakterinin, domuzcuğa ulaşmasını sağlamaz?





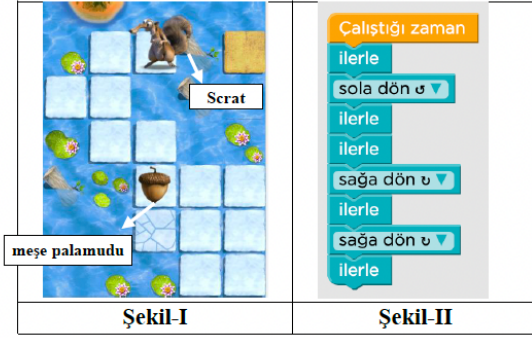
34. Yukarıda Şekil-I'de verilen bulmacada amaç, dört toprak yığını tarladan kaldırmaktır. Şekil-II'de ise bu amaçla hazırlanan kod bloğu seti verilmiştir. Kod bloğu setinde kullanılan "1 çıkarın" kod bloğu, toprak varsa kaldırır yoksa çukur oluşturur. Buna göre program çalıştırıldığında ne olur?

- Çiftçi, tarladaki dört toprak yığını kaldırır ve hiç çukur oluşturmaz.
- Çiftçi, tarladaki dört toprak yığını kaldırır ve üç adet çukur oluşturur.
- Çiftçi, tarladaki üç toprak yığını kaldırır ve dört adet çukur oluşturur.
- Çiftçi, tarladaki toprak yığınlarından hiçbirini kaldırmaz ve yedi adet çukur oluşturur.



35. Yukarıda Şekil I'deki görselde görülen Scrar karakterinin, meşe palamuduna erişmesi için Şekil II'deki kod bloğu seti oluşturulmuştur. Ancak program çalıştırıldığında Scrar meşe palamuduna ulaşamamaktadır. Buna göre programdaki hatalar ayıklandığında program aşağıdakilerden hangisi gibi olmalıdır?

-
-
-
-



36. Yukarıda Şekil-II'de verilen kod bloğu seti çalıştırıldığında, Şekil-I'deki Scrar karakteri meşe palamuduna ulaşır mı?

- Scrar, meşe palamuduna ulaşır. Kodda herhangi bir hata yoktur.
- Scrar, meşe palamuduna ulaşamaz çünkü yanlış yöne döner.
- Scrar, meşe palamuduna ulaşamaz çünkü yeterli "ileri" hareket bloğu yoktur.
- Scrar, meşe palamuduna ulaşamaz çünkü hem yanlış yöne döner hem de yeterli "ileri" hareket bloğu yoktur.

CEVAP ALANI

	A	B	C	D		A	B	C	D
1					19				
2					20				
3					21				
4					22				
5					23				
6					24				
7					25				
8					26				
9					27				
10					28				
11					29				
12					30				
13					31				
14					32				
15					33				
16					34				
17					35				
18					36				

Sınav bitmiştir.

C. Cognitive Load Scale

Bilişsel Yük Ölçeği

Sevgili öğrenciler;

Bu ölçek, Bilişim Teknolojileri ve Yazılım dersinde gerçekleştirdiğiniz kodlama etkinliklerdeki bilişsel yük düzeyinizin belirlenmesi amacıyla hazırlanmıştır. Aşağıda yer alan soruları, bugün Bilişim Teknolojileri ve Yazılım dersinde çözdüğünüz Ders 13-Arı Koşullandırıcılar code.org bulmacalarını göz önünde bulundurarak cevaplandırmanız gerekmektedir. Ölçekte yer alan soruları cevaplandırmanız yaklaşık olarak 5 dakika sürecektir. Hiçbir cümlemin doğru ya da yanlış cevabı yoktur. Verilen ifadeleri dikkatlice okuduktan sonra her biri için “1: Hiç katılmıyorum, 2: Katılmıyorum, 3: Kararsızım, 4: Katılıyorum, 5: Tamamen Katılıyorum” seçeneklerinden size en uygun olanı “X” ile işaretleyiniz. Her ifadeyi yanıtladığınız ve her ifade için sadece bir yanıt alanını işaretlemeniz önemlidir. Aşağıdaki alana ad, ~~soyad~~ ve sınıf bilgilerinizi yazmayı lütfen unutmayınız.

Pınar KEFELİ BERBER
İletişim: pinar.kefeli@erdogan.edu.tr

Adınız ve Soyadınız:	
Sınıfınız:	

Sorular	1	2	3	4	5
	Hiç Katılmıyorum	Katılmıyorum	Kararsızım	Katılıyorum	Tamamen Katılıyorum
1. Etkinliğin kapsadığı konular çok karmaşıktı.					
2. Etkinlik, bana çok karmaşık gelen algoritmaları içeriyordu.					
3. Etkinlik, bana çok karmaşık gelen kavram ve tanımlar içeriyordu.					
4. Etkinliğin yönergeleri ve/veya açıklamaları yeterince açık değildi.					
5. Yönerge ve/veya açıklamalar, öğrenme açısından çok etkisizdi.					
6. Yönerge ve/veya açıklamaların dili tamamen karmaşıktı.					
7. Etkinlik, işlenen konuları kavramama gerçekten katkı sağladı.					
8. Etkinlik, bilgi-işlem/programlama bilgimi ve kavrayışımı gerçekten geliştirdi.					
9. Etkinlik, kapsadığı algoritmaları kavramama gerçekten katkı sağladı.					
10. Etkinlik, kapsadığı kavram ve tanımları anlamama gerçekten katkı sağladı.					

D. Patterns of Adaptive Learning Scale

Uyumsal Öğrenme Örüntüleri Ölçeği

Değerli katılımcı,

Bu anket sizin Bilişim Teknolojileri ve Yazılım dersine yönelik tutum ve hedeflerinizi ölçmeyi amaçlamaktadır. Soruları yanıtlamadan önce dikkatle okuyunuz. Anketteki soruların doğru yada yanlış cevabı yoktur; her soruda size en yakın olan seçeneği işaretleyiniz. Bu anketteki bazı sorular diğerlerine benzemektedir. Bu konuda endişelenmeyin. Tercihlerinizin doğru ya da yanlış olarak bir değerlendirilmesi yapılmayacaktır.

Ankete vereceğiniz cevaplarınız kesinlikle gizli tutulacak, kimseyle paylaşılmayacak ve sadece bilimsel araştırma amacıyla kullanılacaktır. Bu araştırmaya ilişkin tüm bilimsel yayınlarda, “takma isim” kullanılacak olup, bireylere ait herhangi bir isim kullanılmayacaktır.

İfadelere, düşünerek ve içtenlikle vereceğiniz cevaplar için teşekkür ederiz.

Adınız Soyadınız:

Aşağıda Bilişim Teknolojileri ve Yazılım dersinin bir öğrencisi olarak sizinle ilgili bazı sorular yer almaktadır. Aşağıdaki ifadelerin sizin için ne kadar geçerli olduğunu, her madde içerisinde sunulan 5 seçenekten (1-Kesinlikle Katılıyorum, 2-Katılıyorum, 3-Kararsızım, 4-Katılmıyorum, 5-Kesinlikle Katılmıyorum) size en uygun olanı işaretleyerek belirtiniz.	Kesinlikle Katılıyorum	Katılıyorum	Kararsızım	Katılmıyorum	Kesinlikle Katılmıyorum
1. Bilişim Teknolojileri ve Yazılım dersini en iyi şekilde anlamak benim için önemlidir.					
2. Sınıftaki diğer öğrencilerle karşılaştırdığımda zeki görünmek benim için önemlidir.					
3. Sınıftaki diğer öğrencilerin, Bilişim Teknolojileri ve Yazılım dersinde iyi olduğunu düşünmeleri benim için önemlidir.					
4. Bilişim Teknolojileri ve Yazılım dersinde birçok yeni kavram öğrenmek benim için önemlidir.					
5. Bilişim Teknolojileri ve Yazılım dersindeki en zor çalışmaları (alıştırma, <u>ödev</u> , etkinlik,...) bile yapabileceğime eminim.					
6. Sınav sırasında cevapları bazen arkadaşlarımdan alırım.					
7. Bilişim Teknolojileri ve Yazılım dersindeki hedeflerimden biri, öğrenebileceğim en fazlasını öğrenmektir.					
8. Hedeflerimden biri, başkalarına Bilişim Teknolojileri ve Yazılım dersinde iyi olduğumu göstermektir.					
9. Hedeflerimden biri, Bilişim Teknolojileri ve Yazılım dersinde birçok yeni beceri kazanmaktır.					
10. Gayret edersem Bilişim Teknolojileri ve Yazılım dersindeki en zor şeyleri bile yapabilirim.					
11. Sınıf içi Bilişim Teknolojileri ve Yazılım çalışmalarında (alıştırma, <u>ödev</u> , <u>etkinlik</u> ...) bazen kopya çekerim.					
12. Bilişim Teknolojileri ve Yazılım dersindeki hedeflerimden biri, başkalarının, benim zeki olmadığımı düşünmelerini önlemektir.					
13. Bilişim Teknolojileri ve Yazılım dersinde öğretilenleri en iyi şekilde öğrenebileceğime eminim.					
14. Bilişim Teknolojileri ve Yazılım çalışmalarını yaparken bazen cevapları arkadaşlarımdan yazırım.					
15. Hedeflerimden biri başkalarına, Bilişim Teknolojileri ve Yazılım dersinin benim için kolay olduğunu göstermektir.					

16. Hedeflerimden biri, sınıftaki diğer öğrencilerle karşılaştırıldığımda zeki görünmektir.					
17. Bilişim Teknolojileri ve Yazılım dersini anlamıyormuş gibi görünmek istemem.					
18. Becerilerimi geliştirmek benim için önemlidir.					
19. Öğretmenimin, benim sınıftakilerden daha az bildiğimi düşünmemesi benim için önemlidir.					
20. Pes etmezsem, Bilişim Teknolojileri ve Yazılım dersindeki hemen hemen her çalışmayı yapabilirim.					
21. Hedeflerimden biri, Bilişim Teknolojileri ve Yazılım dersinde zorlanıyormuş gibi görünmemektir.					
22. Yapılması veya öğrenilmesi gereken şey zor olsa bile öğrenebilirim.					

Aşağıdaki ifadelerin sizin için ne kadar geçerli olduğunu, her madde içerisinde sunulan 5 seçenekten (1-Kesinlikle Geçerli, 2-Geçerli, 3-Kararsızım, 4-Geçersiz, 5-Kesinlikle Geçersiz) size en uygun olanı işaretleyerek belirtiniz.	Kesinlikle Geçerli	Geçerli	Kararsızım	Geçersiz	Kesinlikle Geçersiz
23. Bazı öğrenciler sınavdan bir gün önce vaktini boşa geçiriyorlar. Sonra da sınavda iyi yapamazlarsa bunu sebep olarak gösteriyorlar. Bu senin için ne kadar geçerli?					
24. Bazı öğrenciler kendi istekleriyle bir sürü etkinliğe katılıyorlar. Sonra da sınıf <u>çalışmalarını</u> iyi yapamazlarsa başka şeylerle uğraştıkları için böyle olduğunu söylüyorlar. Bu senin için ne kadar geçerli?					
25. Bazı öğrenciler ders çalışmamak için sebep arıyorlar (kendini iyi hissetmemek, annesi ve babasına yardım etmek, kardeşine bakmak gibi). Sonra da sınıf çalışmalarını iyi yapamazlarsa, sebebin bu olduğunu söylüyorlar. Bu senin için ne kadar geçerli?					
26. Bazı öğrenciler arkadaşlarının, kendi dikkatini dağıtmalarına ya da ödevlerini yapmasını engellemelerine izin veriyorlar. Sonra da sınıf çalışmalarını iyi yapamazlarsa arkadaşlarının çalışmalarına engel olduklarını söylüyorlar. Bu senin için ne kadar geçerli?					
27. Bazı öğrenciler derste kasıtlı olarak çok gayret etmiyorlar. Sonra da sınıf çalışmalarını iyi yapamazlarsa gayret etmedikleri için olduğunu söylüyorlar. Bu senin için ne kadar geçerli?					
28. Bazı öğrenciler Bilişim Teknolojileri ve Yazılım çalışmalarını yapmayı son dakikaya bırakıyorlar. Sonra da derslerini iyi yapamazlarsa bu yüzden iyi yapamadıklarını söylüyorlar. Bu senin için ne kadar geçerli?					

Aşağıdaki sorular Bilişim Teknolojileri ve Yazılım dersinde sınıfınızın durumu ile ilgilidir. Gerçekten ne hissediyorsanız onu işaretlemeyi unutmayın. Her madde içerisinde sunulan 5 seçenekten (1-Kesinlikle Katılıyorum, 2-Katılıyorum, 3-Kararsızım, 4-Katılmıyorum, 5-Kesinlikle Katılmıyorum) size en uygun olanı işaretleyiniz.	Kesinlikle Katılıyorum	Katılıyorum	Kararsızım	Katılmıyorum	Kesinlikle Katılmıyorum
29. Bizim sınıfta gayret etmek çok önemlidir.					
30. Bizim sınıfta asıl hedef, iyi not almaktır.					
31. Bizim sınıfta asıl hedef, derste islenen konuları gerçek anlamda anlamaktır.					
32. Bizim sınıfta doğru cevap vermek çok önemlidir.					
33. Bizim sınıfta kimse, diğer öğrencilerden başarısız olmak istemez.					
34. Bizim sınıfta diğer öğrencilerin önünde hata yapmamak önemlidir.					
35. Bizim sınıfta ne kadar ilerleme gösterdiğin gerçekten önemlidir.					

36. Bizim sınıfta dersi ezberlemek değil anlamak önemlidir.					
37. Bizim sınıfta diğer öğrencilere, derste başarısız olmadığımı göstermek gerçekten önemlidir.					
38. Bizim sınıfta yeni fikir ve kavramları öğrenmek çok önemlidir.					
39. Bizim sınıfta derse ilgisiz görünmemek çok önemlidir.					
40. Bizim sınıfta eğer bir şeyler öğreniyorsak, yanlış yapmamız önemli değildir.					
41. Bizim sınıfta sınavlardan yüksek not almak çok önemlidir.					
42. Bizim sınıfta kimse dersi anlamıyormuş gibi görünmek istemez.					

E. Attitudes Toward Coding Education Scale

Kodlama Eğitime Yönelik Tutum Ölçeği (KEYTÖ)

Değerli öğrenciler,

Bu ölçek, Bilişim Teknolojileri ve Yazılım dersinde yer alan Kodlama eğitimi ile ilgili görüşleriniz hakkında bilgi edinmek amacıyla hazırlanmıştır.

Kodlama (Bilgisayar programlama); problemleri çözmek, insan-bilgisayar etkileşimini sağlamak ve belirli bir görevi bilgisayarlar tarafından gerçekleştirmek için çeşitli komut setleri ile yapılan uygulama ve geliştirme sürecidir.

Bu ölçekte yer alan maddelerin hiçbirisinin doğru ya da yanlış yanıtı yoktur. Bu ölçeğe bireysel olarak vereceğiniz yanıtlar kesinlikle üçüncü şahıslara ve diğer resmi kurum ve kuruluşlara bildirilmeyecektir. Ölçek sonuçlarının sağlıklı olabilmesi için, soruları samimi ve doğru olarak yanıtlamanız büyük önem taşımaktadır. Yanıtlamaya başlamadan önce cümleyi dikkatlice okuyunuz. Sizden istenen, maddelere 1'den 5'e kadar bir puan vermenizdir.

1-Kesinlikle Katılmıyorum

2-Katılmıyorum

3- Kısmen Katılıyorum / Kısmen Katılmıyorum

4- Katılıyorum

5-Kesinlikle Katılıyorum

Araştırmaya katıldığınız için teşekkür eder, derslerinizde başarılar dilerim.

Öğr. Gör. Pınar KEFELİ BERBER

Recep Tayyip Erdoğan Üniversitesi/Bilgisayar Teknolojileri Bölümü

İletişim: pinar.kefeli@erdogan.edu.tr

Ad-Sovad:

		(1) Kesinlikle Katılmıyorum	(2) Katılmıyorum	(3) Kısmen Katılıyorum / Kısmen Katılmıyorum	(4) Katılıyorum	(5) Kesinlikle Katılıyorum
1	Kodlama eğitimini severim.					
2	Kodlama eğitiminin ne anlama geldiğini bilirim.					
3	Kodlama eğitimi bana çok kolay geliyor.					
4	Birçok dersi yapıyorum ama kodlama konusunda hiç yeteneğim yok.					
5	Kodlama öğrenirken hiç zorlanmıyorum.					
6	Kodlama eğitimini günlük hayatta kullanabileceğimi düşünüyorum.					
7	Kodlama yaparken mutlu olurum.					
8	Kodlama yaparken kendimi mutsuz hissederim.					
9	Kodlama eğitimi sırasında eğlenirim.					
10	Kodlama eğitimi sırasında sıkılırım.					
11	Kodlama eğitimi alırken bir şeyler öğrendiğimi hissederim.					
12	Kodlama yarışmalarında derece almak beni çok mutlu eder.					

Lütfen arka sayfaya geçin.

		(1) Kesinlikle Katılmıyorum	(2) Katılmıyorum	(3) Kısmen Katılıyorum / Kısmen Katılmıyorum	(4) Katılıyorum	(5) Kesinlikle Katılıyorum
13	Kodlama eğitimi bana çok zor geliyor.					
14	Kodlama yaparken kendime güvenirim.					
15	Kodlama eğitimi sırasında hiçbir şey öğrenemiyorum.					
16	Kodlama eğitimi diğer derslerdeki başarı durumumu olumlu etkiliyor.					
17	Kodlama yaparken bir problemle karşılaştığım zaman çözüme ulaşamazsam vazgeçerim.					
18	Kodlama öğrenirken çok zorlanıyorum.					
19	Kodlama eğitiminin ileride işime yarayacağını düşünüyorum.					
20	Kodlama yapmayı bilmek, iş bulma konusunda çok işime yarayacak.					
21	Kodlama eğitiminin olduğu gün okula gitmek istemem.					
22	Kodlama öğrenmeye çalışmak zaman kaybıdır.					
23	Kodlama eğitimi sırasında öğrendiğim bilgileri diğer derslerde de kullanabilirim.					
24	Kodlama eğitimini sevmem.					
25	Kodlama yapmaktan zevk alırım.					
26	Çok uğraşmama rağmen kodlama bana zor geliyor.					
27	Kodlama öğrenmek benim için önemlidir.					
28	Kodlama eğitimi aldığım dersteki yüksek başarı durumum, diğer derslerime de olumlu katkı sağlamaktadır.					
29	Kodlama öğrenmenin ileride işime yarayacağına inanırım.					
30	Kodlama eğitiminin olduğu gün okula mutlu giderim.					
31	Kodlama ile uğraşırken karşılaştığım problemlerin çözümünde kendime güveniyorum.					
32	Kodlama yapmayı "kesinlikle" öğreneceğimi düşünüyorum.					
33	Kodlama eğitimini çok önemsiyorum.					
34	En başarısız olduğum şey kodlama yapmaktır.					
35	İleride karşılaşacağım daha zor kodlama çalışmalarının üstesinden gelebileceğimi düşünüyorum.					
36	Kodlama eğitimi aldığım dersten iyi not alabilirim.					
37	Kodlama yaparken bir problemle karşılaştığım zaman çözüme ulaşana kadar uğraşırım.					
38	Kodlama konusunda büyüklerimden yardım almadan başarılı olabilirim.					
39	Kodlama eğitimi sırasında öğretmenim tarafından örnek gösterilmek beni gururlandırır.					
40	Kodlama eğitimi aldığım dersten yüksek not almak beni mutlu eder.					
41	Kodlama eğitimini hiç önemli görmüyorum.					

F. Reading Comprehension Achievement Test

Adınız-Soyadınız:	
Okulunuz:	
Sınıfınız ve okul numaranız:	

Okuduğunu Anlama Başarı Testi

Sevgili öğrenci,
Okuduğunu anlamaya yönelik olarak hazırlanmış bu testte, farklı okuma parçaları ile ilgili sorular bulunmaktadır. Parçaları dikkatli bir şekilde okuyarak okuduğunuz parçayla ilgili verilen soruları yanıtlayınız. Soruların hepsini yanıtlamaya çalışınız. Her soru için verilen seçeneklerden yalnızca birini işaretleyiniz. Yukarıdaki alana ad-soyad, okul, sınıf ve okul numarası bilgilerinizi yazmayı unutmayınız.
Bu testte aldığınız puanlar ders başarı notunuza etki etmeyecektir ve cevaplarınız araştırmacı tarafından özenle saklanarak kimseyle paylaşılmayacaktır.

“1. METİN”

Başarılı olma yolunda senin ilk büyük düşmanın tembelliktir. Burada sana tembelliği tarif edecek değilim. Onu sen, ben hepimiz az çok tanırız. Yalnız ben sana şunu söyleyeceğim ki, tembellik insan karşısına çıkıp da mertçe savaşan bir düşman değildir. Aksine, eski peri hikayelerindeki kahramanlar gibi, şekilden şekle girerek ve bin bir hile kullanarak alt etmeğe çalışan bir korkaktır. Tehlikesinin büyüklüğü de buradan gelmektedir.

Tembelliğin, yerine, adamına ve çağına göre girmedeği kalıp yoktur. Herkesin huyuna göre tavır alır ve konuşur. Dilimizde aldığı çeşitli isimler, onun bu sinsiliğini gösterir. Tembelliğin adı havailiktir. Bir adı gevşeklik ve hoppalık, diğer bir adı da uyuşukluk, üşengeçlik ve keyfine düşkünlüktür. Tembellik herkesin karşısına her zaman aynı kılıfta çıkmaz. O, mesleksiz aktör gibi her zaman rol değiştirir. Bazen geçerli bir mazeret kılıfına girer; hasta olur, yorgun düşer ve haline acındırır. Bazen tatlı bir dille konuşur, göntül çeler.

Tembelliğin kitabında daha pek çok şey vardır. Yalnız şunu söyleyeyim ki, eğer tembel isen ve tembelliğin organlarla ilgili bir hastalıktan ileri gelmiyor da ruhsal bir gevşeklik, uyuşukluk, üşengeçlik, hoppalık ve havailik şeklinde ise, iradeni kullanmak suretiyle başarının bir düşmanını yenebilirsin.

Anonim

1. “O, mesleksiz aktör gibi daima rol değiştirir.” cümlesi nasıl bir karakterin ifadesidir?

- A) Tutarlı-sinsi-girişken
- B) Tutarsız-değişken-sinsi
- C) Dengesiz-inandırıcı-çalışkan
- D) Tutarsız-güvenilir-değişken

2. Aşağıdakilerden hangisi, tembelliğin tehlikelerinden biri **olamaz**?

- A) Herkesin huyuna göre tavır alıp, konuşması
- B) Bin bir hile kullanarak insanı alt etmeğe çalışması
- C) Tembelliğin, aktör gibi daima rol değiştirmesi
- D) İnsanın karşısına çıkıp, korkmadan savaşması

3. Aşağıdakilerden hangisi bu parçaya **en uygun** başlık olur?

- A) Başarının Düşmanı
- B) Maskesiz Aktör
- C) Başarının Yolu
- D) En Büyük Düşman

4. Aşağıdaki seçeneklerden hangisi tembel bir insanın söyleyeceklerine **ters düşer**?

- A) Sınavlar olmasaydı, hiç ders çalışmazdım.
- B) Şansın varsa her şeyin var demektir.
- C) Emek olmadan yemek olmaz.
- D) Kütüphanede pineklemekten hoşlanmam.

Arka sayfaya geçiniz.

“2. METİN”

Yücel beşinci sınıfa gidiyordu. Zeki, çalışkan bir öğrenciydi. Az önce okuldan döndü. Öğretmeninin verdiği ev ödevlerini yaptı. Sonra gazeteyi alıp okumaya başladı.

Gazetenin ilk sayfasında bir uzay roketinin resmi vardı. Bu, kocaman bir roketti. Üst kısmı sivri, kenarları düzgün yuvarlak. Yücel uzun uzun bu resme baktı, onu evirdi çevirdi. Aklına bir soru takılmıştı. Bu kocaman şey, nasıl oluyor da gökyüzünün uçsuz bucaksız derinliklerine kadar yükselebiliyor? Yeryüzünden nasıl ayrılıyor, havayı nasıl deliyor?

Düşündükçe, Yücel’in merakı daha da artıyordu. Ne yapmalıydı? Bunu nasıl öğrenmeliydi? Aklına bir soru takıldı mı hep böyle oluyordu. Öğrenmeden rahat edemezdi.

Yücel, aklına takılan soruyu ablasına sordu. Fakat sorusuna tam yanıt alamadı. Ablası:

- Git, Zeki amcana sor. O bilir, dedi.

Zeki Bey, Yücel’in kapı komşusuydu. Uzun yıllar pilotluk yapmış, sonra emekli olmuştu. Yücel gazeteyi alıp ona gitti:

- Zeki amca, size bir sorum var, dedi. Resimdeki şu roket, nasıl oluyor da Dünya’mızdan ayrılıp çok yükseklerle kadar çıkabiliyor? Bana, bunu anlatır mısınız?

Zeki amca, biraz düşündükten sonra anlatmaya başladı:

- Bir balonu şişirip havaya bırakalım. Balonun uçuşunu gördünüz. Çünkü balonu şişirdiğimiz zaman, dar bir yere hava sıkıştırmış oluruz. Balonu bırakınca da içindeki hava dışarı çıkmak ister, çıkan bu havanın basıncı, balonu uçurur. İşte roket de böyle uçar.

- Yani rokete de hava mı doldurulur, Zeki amca?

- Hayır, hava doldurulmaz. Rokette itici gücü çok daha kuvvetli bir gazdan yararlanılır.

- Roketin havalanışı nasıl olur? O zaman neler görülür, Zeki amca?

- Bunu sırasıyla anlatacağım. Fırlatılacak roket hazırlanır. Özel giysiler giymiş adamlar, önce rokete gerekli yakıtı koyarlar. Bu tehlikeli bir iştir, yangın çıkabilir. Bunun için orada sürekli bir yangın arabası bekler. Yakıt koyma işi bitince sıra, roketin ateşlemesine gelir. Bu sırada herkes koruma odasına girer. Roketin yanında kimse bulunmaz. Koruma odasında şöyle bir geri sayış başlar: 10-9-8-7-6-5-4-3-2-1. ardından ateş emri verilir.

Roket alevler saçarak yükselir. Gittikçe hızlanır. Gök gürültüsünü andıran bir ses çıkararak hızla gözden kaybolur.

Yücel, roketin nasıl atıldığını, nasıl yükseldiğini öğrenmişti. Zeki amcasına teşekkür ederek oradan ayrıldı.

5. Yücel’in gazetede gördüğü roket nasıldı?

- A) Küçük, üst kısmı sivri, kenarları düzgün ve yuvarlak
- B) Kocaman, üst kısmı düzgün, kenarları yuvarlak
- C) Küçük, üst kısmı sivri, kenarları düzgün
- D) Kocaman, üst kısmı sivri, kenarları düzgün ve yuvarlak

6. Aşağıda karışık olarak verilmiş cümlelerin doğru sıralanışı hangisidir?

- 1.Koruma odasında geri sayış başlar
 - 2.Fırlatılacak rokete gerekli yakıt konur.
 - 3.Roket alev alarak yükselir.
 - 4.Roketin ateşleme işlemine geçilir.
- A) 1, 2, 3, 4 B) 1, 3, 4, 2
C) 2, 4, 1, 3 D) 2, 4, 3, 1

Diğer sayfaya geçiniz.

“3. METİN”

Japonya’da ay ışığını seyretme toplantısı yaparlar, sizi oraya çağırırlar ancak orada hiç konuşulmaz. Güzel ve zevkli bir bahçede oturur, ayın doğmasını seyreder ve bundan zevk almayı öğrenirsiniz.

Japonlar doğaya, bizim anlayamayacağımız kadar hayranlık duyarlar. Kışın ilk yağan karı seyretmek ve kutlamak için toplantılar yaparlar. Karın çevreyi birden nasıl değiştirdiğini ve yumuşattığını, ışık gölge arasındaki farkları ortadan kaldırmasının zevkini tadarlar. Japonlar güzel bir yaz gecesi kirlarda dinlemeğe giderler. Evet dinlemeğe giderler. Neyi biliyor musunuz? Böceklerin müziğini...

Bir gün beni pek çok kibar bayanın mangal gibi bir şeyin çevresinde oturdukları bir toplantıya çağırılmışlardı. Mangal kömürtü yanarken içine değişik odun parçaları atıyor, biraz yanınca kadar içinde bırakıyorlardı. Sonra duman çıkarken bu odun parçalarını, özel bir tepsi içinde herkes koklasın diye, sıra ile gezdiriyorlardı. İnsan ilk kez orada şeftali, çam, kiraz gibi odunların, müzik notaları gibi birbirinden ayrı kokuları olabileceğinin farkına varıyordu.

7. Değişik odun parçalarının “mangalda yakılması” neyi göstermektedir?

- A) Japon kadınlarının sanatçı ruhlu olduğunu
- B) Her bitkinin farklı kokusunun bulunduğunu
- C) Yanan odun parçalarının tepsiyle taşındığını
- D) Yanan odunların farklı kokular çıkardığını

8. Parçada hangi duyumuzla ilgili ayrıntıya yer verilmemiştir?

- A) Görme B) İşitme
- C) Tatma D) Koklama

9. Parçaya göre Japonlar kışın neyi seyrederler?

- A) İlk karın getirdiği değişiklikleri
- B) Değişen çevre koşullarını
- C) Işıkla gölge arasındaki farkları
- D) Yumuşayan insan çizgilerini

10. Parçaya göre “böceklerin müziğini dinlemek” hangi anlamda kullanılmıştır?

- A) Sevilen müzik parçalarını dinlemek
- B) Hayvanları sevmek ve onlarla ilgilenmek
- C) Böceklerin çıkardığı sesleri ayırt etmek
- D) Doğadan gelen sesleri dinlemek

11. Japonya’da ay ışığını seyretme toplantısı düzenlenmesinin **asıl amacı** nedir?

- A) Konukları ile iyi zaman geçirmek
- B) Ayın nasıl doğduğunu göstermek
- C) Ayın doğuşundan zevk almak
- D) Konuklarını bahçede ağırlamak

12. Parçaya **en uygun** başlık aşağıdakilerden hangisidir?

- A) Japon Gelenekleri
- B) Doğa Sevgisi
- C) İlk Düşen Kar
- D) Böceklerin Müziği

Arka sayfaya geçiniz.

“4. METİN”

Bir komşu der ki Hoca'ya:
“Eşeğimi ettim kayıp,
Belki inmiştir çaya,
Sana zahmet arayıp,
Bul da getir çabucağ”
Rahmetli kıyı bucak,
Arayacağı yerde,
Başı havada, tiz perde
Bir şarkı tutturur;
Bahçe, bağ gezip durur.
Biri Hoca'ya rastlar,
Der: “Bağda ne işin var?”

- Eşeği arıyorum.
Dereye varıyorum.
- İşi tutarak gevşek,
Aranmaz böyle eşek,
Bırakır şarkıyı,
Fırıldar köşe kıyı,
Sendeki bu neşeyi,
Gören düğünde sanır.
- El alemin eşeği
Ancak böyle aranır.

Hasan Ali YÜCEL

13. Aşağıdakilerden hangisi bu şiire **en uygun** başlık olur?

- A) Hoca'nın eşeği B) Ancak böyle aranır
C) Hoca'nın neşesi D) Dostluk

14. Eşeğin sahibi kimdir?

- A) Nasrettin Hoca B) Hoca'nın karısı
C) Hoca'nın komşusu D) Hoca'nın akrabası

15. Hocayla karşılaşan kişi Hoca'yı neden eleştiriyor?

- A) Kendisini hiç arayıp sormadığı için
B) Borcunu ödemediği için
C) Üstlendiği işi özenli yapmadığı için
D) Bahçeye izinsiz girdiği için

“5. METİN”

Denizle iç içe olmuş insanlar için bile buzkıranla yolculuk etmek çok garip bir deneyimdir. Denizci, açık sularda, geminin dalgalar boyunca ne şekilde tepki göstereceğini öğrenir. Buzla kaplı bir denizde ise, düzenli hareketler yoktur. Onun yerine, orada gemiciyi hayrete düşüren beklenmedik sarsıntılar vardır.

Dev gibi buz parçaları, çelik geminin gövdesine çarpar. Çarpma sırasında çıkan ses, geminin su seviyesini gösteren çizgi boyunca gürültü çıkaran yüzlerce çöp tenekesinin sesine benzer. Geminin içinde, o anda görevli olmayan gemiciler dinlenmeye çalışır, ancak bu gürültüden kaçmak mümkün değildir.

Gemi buz kırma işlemi başladığında, buzlardan oluşan geniş bir alana girer ve aniden yavaşlar. Sonra geminin çelik gövdesi kalın buz parçasının üzerinde yükselir. Bu yüzden bir süre suyun yüzeyinde hareketsiz kalır. Geminin ağırlığı buzı kırar. Motorlarının bütün gücüyle çalışan gemi tekrar geri döner ve tekrar buzların içine gömülür. Bu işlem, gemi donmuş sular arasından yol bulup hareket edene kadar sürer.

16. Aşağıdakilerden hangisi gemicilerin dinlenmesini engellemektedir?

- A) Gemi motorunun gürültüsü
B) Gemiye çarpan dalgalar
C) Gürültü yapan gemiciler
D) Gemiye çarpan buzlar

17. Aşağıdakilerden hangisi bu parçaya **en uygun** başlık olur?

- A) Gemiciliğin Zorlukları
B) Kuzey Buz Denizleri
C) Bir Gemicinin Anlatıyor
D) Buzkıranla Yolculuk

Diğer sayfaya geçiniz.

“6. METİN”

Bugün yeni eve taşındık. Yeni evimiz çok güzel. Odanın penceresinden deniz görüntüyor. Annem bu oda senin dediğinde, hemen pencereden dışarı baktım. Denizi görünce kendimi kaptan köşkünde sandım. Deniz deyince gemi, gemi deyince kaptan, kaptan deyince de kaptan köşkü gelir aklıma. Adalara sefer yapan, yüzünü hiç görmediğim kaptan gibi ben de köşküme çiçeklerle süsleyeceğim.

İlk benim eşyalarımı odama yerleştirdik. Ortada geniş, boş bir alan kaldı. Eski odam küçüktü. Oyuncaklarım, kitaplar, defterler... hepsi tıksı tıksı. Bu oda geniş. Yeni evimizin büyükçe bir salonu ve mutfak var. Ev ferah. Annem bu yüzden yeni evimizde daha mutlu olacağımızı söylüyor.

18. Parçada anlatılmak istenen **temel düşünce** nedir?

- A) Ev taşınmanın ne kadar zor olduğu
- B) Eski evlerinin ne kadar kötü olduğu
- C) Yeni taşındıkları evin özellikleri
- D) Odaları nasıl düzenledikleri

19. Yeni evde neden daha mutlu olacaklar?

- A) Daha geniş ve ferah olduğu için
- B) Binası yeni yapıldığı için
- C) Yeni komşuları iyi olduğu için
- D) Deniz kenarında olduğu için

20. Çocuk odasına girince neden kendini kaptan köşkünde zannediyor?

- A) Geçen gemilerin siren seslerini işittiği için
- B) Odasının penceresinden deniz gördüğü için
- C) Kendisini adalara sefer yapan kaptanlara benzettiği için
- D) Büyüdüğünde gemilerde kaptan olmak istediği için

“7. METİN”

Bir adam bir eşek satın almak istedi. Hayvan pazarında satılık bir eşek buldu. Satın almadan önce onu denemek için eşek sahibiyle anlaştı. Hayvanı eve götürdü ve diğer eşeklerle birlikte ahıra koydu. Eşek, ahırda diğer eşeklerden ayrıldı ve en değersiz, haylaz olanıyla arkadaşlık etmeye başladı. Adam, yeni eşeğine hemen bir yular taktı ve sahibine geri götürdü. Bu kadar kısa süre içerisinde uygun bir deneme yapılmasına şaşırان eşek sahibine; “Daha fazla denemeye ihtiyacım yok, arkadaş olarak seçtiği eşekten onun ileride nasıl bir eşek olabileceğini tahmin ediyorum” dedi.

21. Eşeği almak isteyen kişinin hangi hareketi eşek sahibini şaşırttı?

- A) Böyle bir eşeği satın almak istemesi
- B) Bir ahır dolusu eşeğe sahip olması
- C) Eşeği bu kadar hızla geri getirmesi
- D) Eşeği hemen denemek istemesi

22. Eşek sahibinin eşeğini hemen satamamasının nedeni aşağıdakilerden hangisidir?

- A) Eşeği almak isteyen kişi ile beraber çalışmak istemesi
- B) Eşeğin denenmek üzere götürülmesine izin vermesi
- C) Eşek için değerinden çok daha yüksek fiyat istemesi
- D) Eşeği almak isteyen kişiyi aldatmayı denemesi

23. Eşeği almak isteyen adam için aşağıdakilerden hangisi **söylenemez**?

- A) Dikkatli
- B) Kurnaz
- C) Akıllı
- D) Cimri

24. Parçada anlatılmak istenen **temel düşünce** nedir?

- A) Bir malı satın alırken dikkatli olmalıyız.
- B) Arkadaşlarımızı seçerken dikkatli olmalıyız.
- C) Denemeden hiçbir şey satın almamalıyız.
- D) Tek bir kişiyle arkadaşlık etmemeliyiz.

Arka sayfaya geçiniz.

“8. METİN”

“İster kağıt üzerinde olsun ister ağızla, benim sevdiğim konuşma düpedüz, içten gelen, tatlı, büyültü ve kısa süren bir konuşmadır. Güç olsun, zararı yok; ama sıkıcı olmasın; süsten, özentiden kaçsın; düzensiz, geliştigüzel ve korkmadan yüttüstün. Dinleyen her yediğı lokmayı tadarak yesin.”

25. “Dinleyen her yediğı lokmayı tadarak yesin.”

Cümlesinde anlatılmak istenen duygu nedir?

- A) Dinlemekten zevk almak.
- B) Dinlemek zorunda olmak.
- C) Dinleme kurallarına uymak.
- D) İyi bir dinleyici olmak.

26. Aşağıdakilerden hangisi yazarın sevdiği konuşma tarzına **girmez**?

- A) Kısa süreli konuşmalar
- B) İçten gelen konuşmalar
- C) Özentili konuşmalar
- D) Geliştigüzel konuşmalar

“9. METİN”

Keçiciğın aklı bir karış havada ya, stirtistintü bir yana bırakmış, bir başına otlaya otlaya çekip gitmiş. Hain koca kurt, kaçırır mı; hemen görmüş keçiciğı: “Hah, işte ağzıma layık bir lokma. Yaşasın!” demiş.

Keçicik, bakmış, can pazarı. Hiç kurtuluşu yok. “Eh ne yapalım, demek kaderimizde sana yem olmak varmış, kurt kardeş.” demiş. “Madem ölüm kapıya geldi, bari bana biraz kaval çal ki neşeleneyim, kendimi unutup öyle öleyim...”

Kurt, “son isteğı zavallının.” demiş, bulmuş bir kaval, çalmaya başlamış. Kurt çalmış, keçicik oynamış. Derken ötelerden kaval sesini duyan köpekler koşturmuşlar, gelmişler, kurdu önlerine düştürtüp bir güzel kovalamışlar. Kaçmadan önce kurt durumu anlayıp oyuna geldiğini sezinlemiş: “Suç sende değil bende. Neme gerekti benim kaval çalmak, neme gerekti bana köçekli kurban!...” demiş.

Zamansız bir işe kalkışmanın sonu budur. Ölçmeli, biçmeli, adımını ona göre atmalı. Ters oldu mu, işte böyle Dimyat’a pirince giderken evdeki bulgurdan olunur.

27. Aşağıdaki cümlelerin hangisinde “can pazarı” sözü parçadaki kullanımıyla aynı anlamdadır?

- A) Bugün can pazarında her şey ucuzdu.
- B) Sizinle can pazarına gelmeyi çok isterdim.
- C) Bizim keçiyi dün can pazarında sattık.
- D) Depremde yaşanan can pazarı bizi üzdü.

28. Bu parçadan çıkaracağımız ders aşağıdaki ifadelerden hangisiyle yakın anlamlıdır?

- A) Av avlanmış, tav tavlammış.
- B) Bin ölçüp bir biçmeli.
- C) Kurt dumanlı havayı sever.
- D) Köpeksiz stürtiye kurt girer.

29. Aşağıdaki kelimelerden hangisi parçada anlatılan keçi için söylenemez?

- A) İnatçı
- B) Kurnaz
- C) Açıkğöz
- D) Sorumsuz

G. 5th Grade Mathematics Achievement Test

Ad-Soyad:	
Numara:	
Sınıf:	

5. Sınıf Matematik Başarı Testi

Sevgili öğrenci,

Elinizdeki testte “Doğal sayılar”, “Doğal sayılarda dört işlem problemleri” alt öğrenme alanlarından oluşan Matematik Başarı Testinin soruları yer almaktadır. Bu sorular sizlerin adı geçen konuları kavrama düzeyinizle ilgili olarak değerlendirilecek olup bu uygulamanın sonuçları hiçbir şekilde notlarınızı etkilemeyecektir. Vereceğiniz yanıtlar kesinlikle gizli tutulacaktır.

Cevaplarınızı hem cevap şıklarının üzerinde hem de arka sayfanın sonunda yer alan cevap anahtarında ilgili sorunun karşısındaki uygun cevap şığına ait kutucuğa “X” işareti koyarak belirtiniz. Her soru için yalnızca bir seçeneği işaretleyiniz. Verilen soruları dikkatli bir şekilde okuyup, anlayarak cevaplandıracağınızdan eminim.

Katkılarınız için çok teşekkür ederim.

1) 560 060 250 sayısının okunuşu aşağıdakilerden hangisidir?

- A) Elli altı milyon altı yüz iki bin elli
B) Elli altı milyon altmış bin iki yüz elli
C) Beş yüz altmış milyon altı bin iki yüz elli
D) Beş yüz altmış milyon altmış bin iki yüz elli

2) 273 TL’yi 540’a tamamlamak için kaç TL’ye ihtiyaç vardır? Problemini gösteren matematik cümlesi aşağıdakilerden hangisidir?

- A) $273+540=?$ B) $273+?=540$
C) $273 \times ?=540$ D) $273 \times 540=?$

3) Bir otobüste 55 kişi vardır. Bunların 32 tanesi bayandır. Durakta yolculardan 3 erkek 7 bayan indiğine göre otobüste kaç kişi kalmıştır? Yukarıda verilen problem cümlesinde hangi bilgi fazla verilmiştir?

- A) Otobüsteki toplam kişi sayısı
B) Durakta inen bayan sayısı
C) Durakta inen erkek sayısı
D) Otobüste bulunan bayan sayısı

4) 19’un iki katı ile 9’un 3 katının farkı kaçtır?

- A) 24 B) 20 C) 38 D) 11

5) Hülya 990 liraya satılan bir bilgisayarı, 120 lira peşin ödeyip geri kalanı 6 eşit taksitle alıyor. Hülya her bir taksit için kaç lira ödeyecektir?

- A) 140 B) 145 C) 150 D) 165

6) 20 sayısının 3 katının 5 fazlası aşağıdakilerden hangisidir?

- A) 60 B) 65 C) 70 D) 75

7) Aşağıdakilerin hangisinde binler bölüğündeki rakamların sayı değerlerinin toplamı, birler bölüğündeki rakamların sayı değerlerinin toplamından büyüktür?

- A) 123 456 987 B) 456 123 987
C) 987 123 456 D) 123 987 456

8) Türkiye’nin 2008 yılındaki nüfusu 8 basamaklı bir doğal sayıdır. Bu sayının birler bölüğünde 100, binler bölüğünde 517 ve milyonlar bölüğünde 70 bulunduğuna göre, 2008 yılında Türkiye’nin nüfusu aşağıdakilerden hangisidir?

- A) 700 517 100 B) 517 070 100
C) 100 517 070 D) 70 517 100

9) 0, 3, 1, 4 rakamlarının tamamı kullanılarak, yazılabilecek en büyük ve en küçük dört basamaklı sayıların arasındaki fark kaçtır?

- A) 2970 B) 3106 C) 3276 D) 4176 84

10) Bir sınıftaki öğrencilere 200 ml’lik kutu sütlerden ikişer tane dağıtılıyor. Bu sınıfta 25 öğrenci olduğuna göre, toplam kaç ml süt dağıtılmıştır?

- A) 400 B) 1000 C) 10000 D) 40000

11) Serap, pazardan kilogramı 3 TL olan elmalardan 6 kg, kilogramı 4 TL olan muzdan 2 kg almıştır. Serap pazarcıya toplam kaç TL para ödeyecektir?

- A) 25 B) 26 C) 27 D) 28

12) Bir öğretmen 96 tane çikolatayı 24 kişilik bir sınıftaki öğrencilere her birine eşit sayıda çikolata düşecek şekilde paylaşmak istiyor. Buna göre her öğrenciye kaç tane çikolata düşmüştür?

- A) 4 B) 6 C) 8 D) 10

13) “Beş yüz iki milyon iki yüz üç” sayısının yazılışı hangisidir?

- A) 502.000.203 B) 50.200.203
C) 500.200.203 D) 500.202.003

14) Bir usta 30 gün çalışmasına karşılık 2400 lira aldığına göre ustanın gündeliği kaç liradır?

- A) 60 B) 70 C) 80 D) 90

15) Aşağıdaki sayılardan hangisinin binler bölümündeki rakamların sayı değerleri toplamı 11’dir?

- A) 281347 B) 280346
C) 282064 D) 181506

16) Atatürk’ün ölüm yılı olan 1938 sayısının basamak değerleri toplamından, doğum yılı olan 1881 sayısının sayı değerlerinin toplamı çıkarıldığında fark kaç olur?

- A) 1918 B) 1920
C) 1922 D) 1923

17) Aylin 8 yıl sonra 18 yaşında olacağına göre dört yıl önce kaç yaşındaydı?

- A) 4 B) 6 C) 8 D) 14

18) 15 in 3 katının 11 eksiği kaç eder? Problemini ifade eden matematik cümlesi aşağıdakilerden hangisidir?

- A) $15+3-11$ B) $15/3-11$
C) $15 \times 3-11$ D) $15 \times 11-3$

19) 5 tanesi 350 TL olan sabunların 45 tanesi kaç liradır?

- A) 1 750 B) 2 250
C) 3 150 D) 4 000

20) 685 sayısının 30 fazlasının 25 eksiği nedir? Yukarıda verilen sorunun cevabı hangisinin cevabı ile aynıdır?

- A) 685 sayısının 10 fazlası
B) 685 sayısının 5 fazlası
C) 685 sayısının 5 eksiği
D) 685 sayısının 30 fazlası

21) Ahmet, pazartesi günü 850 metre yol yürütmüştür. Salı günü ise pazartesi günü yürüdüğü yolun yarısını yürütmüştür. Buna göre Ahmet iki günde toplam kaç metre yol yürütmüştür?

- A) 1200 B) 1275
C) 1400 D) 1550

22) Hangi sayının 50 katı 150’dir?

- A) 2 B) 3 C) 4 D) 5

23) Hakan’ın boyu Kemal’ den 8 cm kısa, Orhan’dan 5 cm uzundur. Orhan’ın boyu 135 cm olduğuna göre üçünün boy uzunlukları toplamı kaç santimetredir?

- A) 439 B) 423
C) 412 D) 399

24) Bir sınıftaki 38 öğrencisinin her birine altışar çikolata veren öğretmenin 16 çikolatası kalıyor. Öğretmenin sınıfa getirdiği çikolatalar kaç tanedir?

- A) 158 B) 200
C) 230 D) 244

25) Bir tavuk çiftliğinde günde 5640 tane yumurta üretilmektedir. Bu yumurtaların her biri 20 yumurta alan kolilere yerleştirilerek satıldığına göre bu çiftlikte 10 günde kaç paket yumurta üretilmiştir?

- A) 2380 B) 2640
C) 2760 D) 2820

CEVAP ANAHTARI

	A	B	C	D		A	B	C	D
1					16				
2					17				
3					18				
4					19				
5					20				
6					21				
7					22				
8					23				
9					24				
10					25				
11									
12									
13									
14									
15									

H. Interview Protocol

Interview Protocol for Students

Görüşme tarihi:

Görüşülen Kişi:

Görüşme saati:

Yer:

Merhaba,

Ben Pınar KEFELİ BERBER, Orta Doğu Teknik Üniversitesi Bilgisayar ve Öğretim Teknolojileri Eğitimi Anabilim Dalında doktora öğrencisiyim. Aynı zamanda Recep Tayyip Erdoğan Üniversitesi'nde Bilgisayar Teknolojileri Bölümü'nde öğretim görevlisiyim. Öncelikle bana vakit ayırarak bu görüşmeyi kabul ettiğin için teşekkür ederim. Bu görüşme ortaokul öğrencilerinin kodlama başarılarını etkileyen faktörlerin belirlenmesine yönelik yürütülen bir çalışma kapsamında gerçekleştirilmektedir. Bu amaçla Bilişim Teknolojileri ve Yazılım dersinde aldığın kodlama eğitimine yönelik bazı sorular sormak istiyorum. Soracağım sorular bilgi ölçme amaçlı değildir. Sadece, senin konu ile ilgili düşüncelerini öğrenmek istiyorum. Araştırma süresince kişisel bilgilerin gizli tutulacaktır. Bu görüşme sadece araştırmacılar tarafından incelenerek sadece bu araştırma için kullanılacaktır. Görüşme sonunda değiştirmek ya da kayıt dışı tutmak istediğin herhangi bir şey olursa bunları değiştirebilir ya da silinmesini isteyebilirsin.

Görüşme ortalama 15-20 dakika sürecektir. Eğer iznin olursa görüşmeyi ses kayıt cihazı ile kaydetmek istiyorum.

Görüşmeye başlamadan önce sormak istediğin herhangi bir soru var mı?

Hazırsan görüşmeye başlamak istiyorum.

Sorular

1. Bu dönem boyunca Bilişim Teknolojileri ve Yazılım dersinde kodlama ile ilgili neler öğrendin? Kısaca özetler misin?
 2. Kodlama ile ilgili en çok hangi kavramları ve konuları öğrenmede/hangi konularla ilgili etkinlikleri tamamlamada/code.org bulmacalarını çözmede zorlandın?
 - o Sence neden zorlandın?
 3. Code.org kodlama platformunun kodlama öğrenmen açısından olumlu yönleri nelerdi? Olumsuz yönleri nelerdi?
 4. Bu dönem boyunca sınıfta gerçekleştirdiğiniz bilgisayarlı etkinliklerin kodlama öğrenmen açısından olumlu yönleri nelerdi? Olumsuz yönleri nelerdi?
 5. Kodlama öğrenirken Bilişim Teknolojileri öğretmeninin sana nasıl yardımcı olduğunu düşünüyorsun?
 6. Dönem boyunca kodlama öğrendiğin derslerde en çok sevdiğin şeyler nelerdi? Neden?
 7. Dönem boyunca kodlama öğrendiğin derslerde en az sevdiğin şeyler nelerdi? Neden?
 8. Kodlama etkinliklerini genellikle bir arkadaşınla birlikte eşli mi yoksa bireysel olarak mı gerçekleştirdin?
 - o Bu senin tercihin miydi?
 - o Sen hangisini tercih ederdin? Neden?
- (Eşli kodlama yapan öğrenciler için)
9. Kodlama etkinliklerinde birlikte çalışacağın grup arkadaşını nasıl seçtin?
 - o Her derste aynı kişiyle mi grup arkadaşı oldun?

10. Dönem boyunca eşli programlama yaparken daha çok hangi rolü üstlendin: Sürücü/Yönlendirici?
 - Bu senin tercihin miydi? Neden?
 - Bilgisayardaki etkinlikler esnasında eşinle görevlerinizi değiştirdiniz mi?
 - Görev değişimini hangi aralıklarla gerçekleştirdiniz?
 11. Bir arkadaşınla birlikte kodlama yapmanın sana göre olumlu yanları nelerdi?
 - Grup arkadaşının, senin kodlama öğrenmene katkı sağladığını düşünüyor musun? Ne şekilde katkı sağladı?
 - Senin, grup arkadaşının kodlama öğrenmesine katkı sağladığını düşünüyor musun? Ne şekilde katkı sağladın?
 12. Bir arkadaşınla birlikte kodlama yapmanın sana göre olumsuz yanları nelerdi?
- (Bireysel kodlama yapan öğrenciler için)*
13. Bireysel kodlama yapmanın sana göre avantajları nelerdi?
 14. Bireysel kodlama yapmanın sana göre dezavantajları nelerdi?

(Ortak)

15. Programlama etkinliklerinde takıldığında, sorunun çözümünü bulamadığında genellikle ne yapıyordun?
 - Arkadaşın/öğretmenin sana ne şekilde yardımcı oluyordu?
16. Kodlama konusunda başarılı olduğunu düşünüyor musun? Neden başarılı olduğunu düşünüyorsun?
17. Kodlamadaki başarını nelerin etkilediğini düşünüyorsun?
18. Kodlama öğrenmek senin için önemli mi? Neden?
19. Öğretmenin kodlama öğretimini daha iyi yapılabilmesi için tavsiyede bulunmak istesen ne söylerdin?

Kodlama eğitimi ile ilgili olarak eklemek istediğin herhangi bir şey var mı?

Teşekkür ederim.

I. Sample Lesson Plan

7. Scrat ile Hata Ayıklama (3. Hafta)	
Önerilen Ders Süresi: 1 ders saati	
Tahmini Hazırlık Süresi: 10 dk.	
Kazanımlar	BT.5.5.1.16. Bir algoritmayı test ederek hataları ayıklar. BT.5.5.2.10. Farklı yapılar için oluşturduğu algoritmaların sonucunu yordayarak hatalarını ayıklar.
Anahtar Kelimeler:	- Yazılım Hatası (Bug) – Düzgün çalışmayan bir programın parçası. - Hata ayıklama – Bir algoritma veya programdaki sorunları bulma ve düzeltme.
Hazırlık	- Sınıfınız için sorun oluşturabilecek alanları belirlemek için Kurs F Çevrimiçi Bulmacalar – 2018 – Web sitesinde yer alan Ders 4: Scrat ile Hata Ayıklama bulmacalarını çözün. (İsteğe bağlı) Sınıfınızla birlikte çözmek için birkaç bulmaca seçin. “Bilgisayar Bilimi Temelleri Ana Aktivite İpuçları – Ders Önerileri”ni inceleyin. (bkz. s. 44) - Hata Ayıklama Yöntemi – Öğrenci Dokümanını sınıfla birlikte gözden geçirin.  Kurs F – Ders 4 Scrat ile Hata Ayıklama Bulmacaları
Giriş	Öğrencilerden günlük yaşamda çözmeleri gereken problemleri düşünmelerini isteyin. - Çalışmayan bir şeyi nasıl düzeltirsiniz? - Belirli bir dizi adımı takip ediyor musunuz? - Bu ünitedeki bulmacalar zaten sizin için çözüldü (oley!), ancak çalışıyor gibi görünmüyorlar (yaaaaaa!) - Programlardaki bu sorunları “yazılım hataları (bugs)” olarak adlandırıyoruz ve burada “hata ayıklamak” sizin işiniz olacak. Hata ayıklama bir süreçtir. İlk önce, programınızda bir hata olduğunu anlamalısınız. Ardından hatayı bulmak için programınızı adım adım ayrıntılı bir şekilde incellersiniz. “İlk adımı dene, işe yaradı mı? Sonra ikinci, peki ya şimdi?” Her şeyin satır satır çalıştığından emin olursanız, kodunuzun gerektiği gibi çalışmadığı bir noktaya geldiğinizde, bir hata bulduğunuzu bilirsiniz. Hatanızı keşfettikten sonra, düzeltmek (veya “hata ayıklamak”) için çalışabilirsiniz! Sınıfta heyecan yaratacağını düşünüyorsanız, bugünkü bulmacalarda yer alan çizgi film karakterini, yani Buz Devri çizgi filminden Scrat’ı tanıtabilirsiniz. Eğer öğrenciler Scrat’ı bilmiyorlarsa, ilginç sincapların başının derde girdiğini gösteren bir video izletin.

Etkinlik	<u><i>Ana Etkinlik (20 dk.)</i></u> <u>Kurs F Çevrimiçi Bulmacalar – 2018 – web sitesi</u> Öğrencilerin bilgisayarda bulmacaları çözmeye başlamalarına izin vermeden önce, onlara Eşli Programlama – Öğrenci Videosu’nda (link: Pair Programming – Student Video) anlatıldığı şekilde eşli programlamanın avantajlarını hatırlatın ve grup arkadaşlarından yardım istemelerini söyleyin. Öğrencileri çiftler halinde oturtun ve yardım için size başvurmadan önce en az iki sınıf arkadaşından yardım istemelerini tavsiye edin.
Ders İpucu	Öğrenci seviyesine göre yukarıdaki bulmacalar yerine Kurs 2 Ders 11 Sanatçı: Hata Ayıklama bulmacalarını çözdürebilirsiniz.

J. Approval of Human Subjects Ethics Committee at METU - I

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
www.ueam.metu.edu.tr

Sayı: 28620816 /

20 Mayıs 2021

Konu : Değerlendirme Sonucu

Gönderen: ODTÜ İnsan Araştırmaları Etik Kurulu (İAEK)

İlgi : İnsan Araştırmaları Etik Kurulu Başvurusu

Sayın Prof.Dr. Soner YILDIRIM

Danışmanlığımı yürüttüğünüz Pınar KEFELİ BERBER'in "Çocuklara Programlama Temellerinin Öğretimine Yönelik Bilişsel Tabanlı Bir Öğretim Tasarımı Model Önerisi" başlıklı araştırmanız İnsan Araştırmaları Etik Kurulu tarafından uygun görülmüş ve **221-ODTU-2021** protokol numarası ile onaylanmıştır.

Saygılarımızla bilgilerinize sunarız.

Dr.Öğretim Üyesi Ali Emre TURGUT
İAEK Başkan Vekili

K. Approval of Human Subjects Ethics Committee at METU - II

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
www.ueam.metu.edu.tr

Sayı: 28620816 /

14 NİSAN 2022

Konu : Değerlendirme Sonucu

Gönderen: ODTÜ İnsan Araştırmaları Etik Kurulu (İAEK)

İlgi : İnsan Araştırmaları Etik Kurulu Başvurusu

Sayın Prof.Dr. Soner YILDIRIM

Danışmanlığını yürüttüğünüz Pınar KEFELİ BERBER'in "Çocuklara Programlama Temellerinin Öğretimine Yönelik Bilişsel Tabanlı Bir Öğretim Tasarımı Model Önerisi" başlıklı araştırmanız İnsan Araştırmaları Etik Kurulu tarafından uygun görülmüş ve **230-ODTÜİAEK-2022** protokol numarası ile onaylanmıştır.

Saygılarımızla bilgilerinize sunarız.

Prof.Dr. Mine MISIRLISOY
İAEK Başkanı

L. Approval of Provincial Directorate of National Education



T.C.
RİZE VALİLİĞİ
İl Millî Eğitim Müdürlüğü

Sayı : E-57774812-605.01-47594595
Konu : Tez Çalışması İzni

11.04.2022

VALİLİK MAKAMINA

İlgi : a)Recep Tayyip Erdoğan Üniversitesi Ardeşen Meslek Yüksekokulunun 04/04/2022 tarihli ve 4940 sayılı yazısı.
b) Millî Eğitim Bakanlığının 21/01/2020 tarihli ve 1563890 (2020/2) sayılı Genelgesi.

Recep Tayyip Erdoğan Üniversitesi Ardeşen Meslek Yüksekokulu öğretim elemanlarından Öğr. Gör. Pınar KEFELİ BERBER'in " Ortaokul Öğrencilerine Programlama Temellerinin Öğretimine Yönelik Bilişsel Tabanlı Bir Öğretim Tasarımı Modeli Önerisi" konulu bilimsel araştırması kapsamında ekte sunulan form ve testleri 2021-2022 Eğitim Öğretim Yılında ilimiz bulunan Resmi Ortaokul Öğrencilerine Programlama Temellerinin Öğretimine Yönelik Bilişsel Tabanlı Bir Öğretim Tasarımı Modeli Önerisi uygulama isteği ilgi yazı ile bildirilmektedir.

Söz konusu form ve testlerin 2021-2022 Eğitim Öğretim Yılında denetimi okul idaresinde olmak üzere, tüm salgın tedbirlerine uyularak, kurum faaliyetlerini aksatmadan, gönüllülük esasına göre ilimiz Resmi Ortaokul Öğrencilerine Programlama Temellerinin Öğretimine Yönelik Bilişsel Tabanlı Bir Öğretim Tasarımı Modeli Önerisi uygulanması Müdürlüğümüzce uygun görülmektedir.

Makamlarınızca da uygun görülmesi halinde olurlarınıza arz ederim.

Ahmet GÜRBÜZ
Müdür a.
Müdür Yardımcısı

OLUR

Engin EMEN
Vali a.
Millî Eğitim Müdürü

Adres : İl Millî Eğitim Müdürlüğü RİZE

Telefon No : 0 (464) 280 53 00
E-Posta: rizemem@meb.gov.tr
Kep Adresi : meb@hs01.kep.tr

Bu belge güvenli elektronik imza ile imzalanmıştır.

Belge Doğrulama Adresi : <https://www.turkiye.gov.tr/meb-ebys>

Bilgi için: Hamit GÖMLEKSİZ

Unvan : Şef

İnternet Adresi: rize.meb.gov.tr

Faks:4642805316

Bu evrak güvenli elektronik imza ile imzalanmıştır. <https://evraksorgu.meb.gov.tr> adresinden 9ff9-a850-337d-bc95-c989 kodu ile teyit edilebilir.

M. The Original Turkish Versions of the Quotes

Theme 1: Cognitive Demands

Inherent Complexity of Programming Concepts and Tasks

Managing Iterative Logic

[S22 in Turkish]: Mesela bir karakter oradayken mesela iki kere o döngüyü kullanmak biraz zor geliyor bana. Mesela üç kez bir şeyi tekrar ettikten sonra mesela üst katmanda bir tane daha şey, döngü koyduğumuz zaman, mesela beş kere olduğu zaman biraz böyle garip bir şeyler oluyordu. Anlayamıyordum pek.

[S6 in Turkish]: Hocam ben bunların zombiyi şeye ulaştırmada çok zorlandım, zombiyi ayçiçeğine ulaştırmak zorluydu. Çünkü hocam diğer çiçekler de var ya vahşi çiçekler, onlardan bir de kaçman gerekiyor. Çünkü hocam kırık buzlar oluyor ya, dikkatsizliğine geliyor ona basıyorsun, düşüyorsun zaten. Ondan kod boşa gidiyor.

Limited Code Blocks Challenges

[S6 in Turkish]: Code.org güzeldi aslında ama bazen bir tane kullanma hakkımız olmasını düşündürmek gerekiyor insanı. Onu ayarlamak zor değil normalde kolay da nerede koyacağını düşünmek gerek aslında.

Sequencing and Logical Flow Difficulties

[S18 in Turkish]: ...Kodları karıştırıyordum. Diğerini yanlışlıkla diğerinin yerine koyuyordum. Karışıyordu.

Diagramming Programming Logic Difficulties

[S27 in Turkish]: İçine mesela... Başla ile başlayınca aklıma soru gelmiyordu ne ile başlayayım falan gelmiyordu. Zorlanıyordum. Hangi komutu nasıl yazmalıyım?

[S27 in Turkish]: Sorularda aklım karışmıştı. Böyle soru sorma değil de üçgen vardı, paralelkenar falan vardı. Orada biraz zorlanıyordum. Karışıklık yapıyordu. ...Neyi, hangi şekli koyacağım bilemiyordum.

Spatial Reasoning Challenges

[S19 in Turkish]: Ben açılar konusunda hiç iyi değilim çünkü matematikte de açılar konusunda hiç iyi değilim. O yüzden bilişime (Bilişim Teknolojileri ve Yazılım dersi) de yansıyor.

[S24 in Turkish]: Orada şekillerde var mesela 180 ilerle. Onu sen 120 edeceksin ya da 145 edeceksin. Orada kararsız kalıyordum ne etsem ya 145 mi, 120 mi, 100 mü yapsam? O yüzden hep denemek gerekiyordu.

Comprehending Code Blocks Functionality

[S20 in Turkish]: Nasıl kullanacağımı anlamadım.

[S12 in Turkish]: En çok son yaptığımız derste yaptığımız bloklarda zorlandım çünkü bilmediğim bloklar vardı. Bilmediğim için yani kodları. Onları kullanmayı zar zor öğrendim yani.

Integration of Multiple Concepts

[S16 in Turkish]: Mesela bir şeyi, zombiyi şeye götür diyordu ya, işte onlarda fazla zorlandım çünkü sağa mı gidecek, sola mı gidecek. Sağa dönüyor. İşte çok fazla blok olduğu zaman ben çok zorlanıyordum. Mesele şöyle yapıyordum dönüyordum kuşun yerine, ne tarafa gidecek o tarafa şey yapıyordum. Sonra da kafam karışıyordu ve yavaşlamaya başlıyordum.

Instructional Factors

Unclear Task Instructions

[S17 in Turkish]: Ve o canlı olmayanları, bilgisayardan olmayanları kafamı karıştırdı. O bardak çok kafamı karıştırdı, beynim yandı.

Abstract Concepts and Confusing Explanations

[S26 in Turkish]: Değişkenlerde mesela şey öğretmen bana şey beş parmak hiç değişmez diyor ama ben böyle yapınca on parmak oluyor, bence değişir diyorum.

Orada birazcık tartışma oldu. Ondan sonra ben çok sıkıntı yaşamaya başladım. Orada birazcık şaşırdım.

[S21 in Turkish]: Yani bana hep şey geliyor... Değişken mesela hepsi değişebilir gibi geliyor. Ama bu şeye bağlı yani oradaki bulmacaya bağlı. Bulmaca öyleyse sabit oluyor. Ama yani değişken de olabilir. ... Sabitin temeli aslında şeye bağlı, bu da benim bir garip düşüncem yani, sabitin temeli oradaki programa bağlı yani. Ne kurduysa sabit o oluyor. Ama ona bakarsak her şey değişebilir. O da benim anlamamı zorlaştırıyor.

Unstructured Learning

[S9 in Turkish]: Bilmiyorduk ne kadar nektar olduğunu. O çok zordu böyle, eğer nektar varsa nektarı al, ilerle, onu yaptırıyorduk. Onda zorlanmıştık. Yani, şey oluyordu böyle bir tane kare vardı, her tarafta nektar vardı böyle. Arıyı ilerleteceğiz mi, nektarı mı aldıracağız diye şaşıırıyordum.

Unsuitable Scaffolding

[S3 in Turkish]: İç-içe döngülerde de şöyleydi genelde. Code.org üzerinden en başta eğitim görürken çok düz oynuyorduk: 4 adım ileri git veyahut da... İlk başta tekrarlanan döngüleri öğrendik. Çok fazla yazmak yerine daha kolay bir şekilde tekrarlanıyordu. Öğrenmek çok fazla zor değildi. Ama iç-içe döngüler bir anda zorlaştı yani.

Time Constraints

[S3 in Turkish]: Daha fazla haftada ders... Haftada iki ders olmuyor. Bir konu üzerinde daha fazla durulabilir. Mesela iç-içe döngülere geçildiğinde sorunla karşılaştığım bazen kafamda çözemediğim noktalar olmuştu. Ya da değişkenlerde... İşte mesela değişkenlerde hala yapamadığım yerler vardı. Konunun üzerinde daha fazla durulabilirdi. Daha iyi olurdu, haftada daha fazla ders olsaydı.

Learning Environment Challenges

Access and Equity

[S13 in Turkish]: Bir de bilgisayarlar çok bozuluyor, onu değiştirmek isterdim.

[S9 in Turkish]: Kodlama ile ilgili çalışmadım. Çünkü... Çalışmadım yani. Bilgisayarım yok.

Login issues

[S19 in Turkish]: Nasıl desem? Şifreli olması biraz zor gibi geldi. Yani kağıdımızı, şifreyi unuttuğumuz zaman şifresiz giremeyebiliyoruz.

Foreign language-related problems

[S5 in Turkish]: Mesela Kurs F'de on üçüncü derste falan ben Türkçe yapmama rağmen dili orada İngilizce konuşturmamız gerekiyor.

[S22 in Turkish]: Şey hani alttaki videolar vardı ya, onları Türkçeye çevirseydi daha iyi olurdu ama.

Theme 2: Effective Instructional Approaches

Plugged Activities

Facilitated Learning

[S9 in Turkish]: Öğrenmemi kolaylaştırdı, büyük bir katkı sağladı. Onu yaparak böyle alıştım, çok hızlı yapmaya başladım. Kendimi geliştirdim kodlamada yani yardımcı oldu bana code.org. Çünkü çözüyorum böyle parçaları birleştirerek yaptığım için daha kolay oluyor. Kodlamayı daha iyi öğreniyordum böyle.

Learning by Doing

[S16 in Turkish]: Etkileri çok fazla oldu çünkü bir öğretmen tahtada çizerek anlatsaydı hiçbir şekilde anlamazdım. Ama Code.org daha kolay geldi bana. O olmasaydı, hoca çizerek anlatsaydı mesela yine anlardım ama az anlardım.

[S30 in Turkish]: Ama kendimiz yaptığımızda daha kaliteli oluyor bence.

Debugging Tasks

[S9 in Turkish]: Mesela kendisi, code.org, şeylerini oluştuyordu, bana derecelerini falan soruyordu. O da işimi kolaylaştırıyordu. Avantajı o oluyordu. Bazen daha kolay oluyordu. O birleştiriyordu parçaları, ben de derecelerini yapıyordum, doksan derece falan...

Rich Content

[S3 in Turkish]: Code.org sayesinde daha detaylı öğrendiğimi düşünüyorum, bazı konularda. Mesela yeni geçenlerde değişkenleri öğrendim ve gerçekten değişkenlerde zorlandım. Bu konuda çok öğretici oldu benim için. Dediğim gibi Scratch'de önceden çoğu terimi biliyordum veyahut da çoğu blok terimlerini. Ama şu iç-içe döngülerde ve değişkenlerde daha iyi öğrendiğimi düşünüyorum. Scratch'de de değişkenler üzerinde duruluyordu ama bu kadar detaylı durulmuyordu. Değişkenleri kesinlikle daha detaylı öğrendiğimi düşünüyorum. Gerçekten de iyi.

Opportunity for Revision and Mastery

[S19 in Turkish]: Çünkü bilgisayarda görüyoruz, yanlışlarımızı görebiliyoruz hemen. Ama kağıt üzerinde yanlışlarımızı göremiyoruz. Yani dediğim gibi bilgisayarda yanlışlarımızı görüyoruz ve ona göre yanlışlarımızı düzeltebiliyoruz. Doğrularımız varsa bir daha gözden geçiriyoruz, yanlış olabilir falan.

Permanent Learning

[S21 in Turkish]: Daha kalıcı olur diye düşünüyorum. Çünkü o anlatım yani bir yere kadar... Zaten sınıf ortamında da öyle iyi anlamak yani öyle tanımlamak o iş zor sınıf ortamında. O yüzden yani burada yapmak daha kalıcı oluyor.

Unplugged Activities

Introduction and Orientation

[S19 in Turkish]: İlk önceden, birinci saat akıllı tahtadan gösterdi. Akıllı tahtadan biz yapmaya başladık. Yanlışlarımızı düzeltmeye başladı. Sonra bilgisayarlardan tek başımıza yapmaya çalıştık. Yani dediğim gibi ilk önce akıllı tahtadan bize öğretiyor,

anlatıyor. Sonra biz yapıyoruz. Yani böyle yapsa güzel olur yani böyle devam etmek isteriz.

Active Engagement

[S6 in Turkish]: Hocam bence en çok sınıfta ettiğimiz daha öğreticiydi. Çünkü bilgisayarda sadece mouse (fare) ve şeyi oynatıyorsun. Ama sınıfta ettiğimiz, kendin hareket ediyorsun, kendin ayarlıyorsun eşyaları.

Real-World Relevance

[S8 in Turkish]: Yani bunu gerçek hayatta da kullanabildiğimizi öğrendim kodlamayı yani komutları.

Blended Approaches

Blending Traditional and Digital Methods

[S13 in Turkish]: Çünkü hocam hem yazıyoruz hem okuyoruz. Yani kendi elimizle yazıyoruz, okuyoruz. ...Bilgisayarda mesela bir tuşa basıyorsun hani şey yapamıyorsun tam. Yaptığımız şeyleri bir kağıda yazmak... Çünkü hocam kağıtta gösterince hem okumuş oluyorsun, ikinci kez okumuş oluyorsun hem de onu mesela dosyasına koysun, unuttuğu zaman oraya bakıp yapsın. Bilgisayarda hani bakamayacak, gidecek, kaydetme diyecek, gidecek.

Teacher Effectiveness

Clear and Effective Explanations

[S26 in Turkish]: Öğretmenim çok güzel kodlama öğretiyor. Anlatması çok etkili.

Supportiveness

[S24 in Turkish]: Sınıf kalabalık, 28 kişi. 28 kişiyle de ilgileniyor. Mesela hani ben dedim ya yapamadığımda sinir oluyorum, o zamanlar hocayı çağırıyorum. Hoca bana anlatıyor öylece kolayca geçiyorum o bölümü. Yani iyi katkı oluyor.

Theme 3: Collaborative Learning Approaches

Pair Selection Criteria

Skill and Expertise

[S17 in Turkish]: Bilgisayar ustasıdır. Öyle kolayları yapamaz. Zeki bir çocuk olduğu için onu seçtim. Akıllı.

Social Compatibility

[S19 in Turkish]: Çünkü o arkadaşımın daha iyiyim. Yani evlerimiz yan yana. Daha iyi ilişkim var onunla. Hem daha iyi arkadaşlığımız var onunla. İlk önceden ben başka bir arkadaşımın oturuyordum. Ondan sonra onunla arkadaşlığımız bitti yani küstük birbirimize. O yüzden hocadan izin aldık ve onunla oturmaya başladım.

Role-Sharing Strategies

Imbalanced Turn-Taking

[S14 in Turkish]: Genelde klavyeyi o kullandı, mouseu ben kullandım. Genelde böyle devam etti.

Regular Turn-Taking

[S16 in Turkish]: Genellikle arkadaşımın sıra sıra oynuyorduk. Birden dokuza kadar olan seviyelerde ben birini oynuyordum, üçünü oynuyordum. Şey, ben biri oynuyorum, o ikiyi oynuyor, ben üçü oynuyorum, o dördü oynuyor.

Benefits of Collaborative Learning

General Positive Perceptions

[S14 in Turkish]: Arkadaşımın oturmak isterdim

Shared Responsibility

[S4 in Turkish]: Daha güzel oluyor. Yani böyle tek başına yapmak yerine arkadaşınla değişimli kullanıyorsun. O yaparken sen onu izliyorsun, sen yaparken de o da seni izliyor, o da olabiliyor. Hem de arada sırada arkadaşın da dinlenmiş

oluyor yazdıktan sonra veya kullandıktan sonra, sen de dinlenmiş oluyorsun, güzel oluyor.

Enhanced Problem-Solving

[S7 in Turkish]: Çünkü daha kolay oluyor. İkimiz birden çözdüğümüz için daha kolay oluyor. Hem ikimiz de yapamadığımız şeyi, mesela birimiz biliyor birimiz bilmiyoruz, öyle görebiliyoruz.

[S21 in Turkish]: ... bir şeye mesela arkadaşın buna bir yönden bakıyor, sen buna diyorum ki bir de bu yönden bakalım, yani çoklu bakış. Bu mantıktan ilerlemek gerekiyor diyor, mesela sen farklı bir mantık şey yapıyorsun. Farklı bir bakış açısı oluyor. Çözemediğiniz sorularda bakış açını değiştirmen gerekiyor probleme.

Mutual Learning and Knowledge Sharing

[S16 in Turkish]: Bence öğrendi ama... Yani mesela, ilk önce başladığımızda sağ-solu pek anlayamıyordu. Ben ona anlatmıştım. Ben şeyleri karıştırıyordum, hani bu şeyi beş kez tekrarlar gibi şeyleri karıştırıyordum. Onu da bana o öğretmişti.

[S26 in Turkish]: Genelde code.org olarak ben ona çok fazla şey öğrettim ama onun dışında bilgisayarın temel şeyleri olarak o bana çok şey öğretti.

Challenges of Collaborative Learning

Unequal Participation

[S17 in Turkish]: Bilgisayarın bana hâkimiyetini vermediği için, artık vermiyor. Dinleniyorum (gülme). Yine böyle yaslanıyorum arkama. Yaptığı şeylerden örnek alıyorum. Bazen de ben yapabileyim diye şey yapıyor, bazen izin veriyor.

Reduced Engagement

[S16 in Turkish]: Bazen hani şey oluyordu u.. o yaparken ben kalıyordum öyle, bakmıyordum ona da. Yardım istediğinde yardım ediyordum ama genellikle onun yaptığı sorulara bakmıyordum.

Conflicts Over Problem-Solving Approaches

[S21 in Turkish]: Bazen tabi ki şey oluyor, bir soru var, çözümü şu diyoruz. O başka bir şey diyor ve bunda inat ediyor inat. Mesela böyle olmalı tamam demiyor kimse, bunu yapalım demiyor, sonra olmazsa benimkisi deneriz demiyor. Ben de yaptım bunu. Oluyor bazen.

Conflicts Over Resource Sharing

[S12 in Turkish]: En az sevdiğim şey bilgisayarı paylaştığım arkadaşımın bilgisayarı paylaşma sorunu yaşadığım için.

Seeking Assistance

Source of Assistance

Peers or Teachers as a Source of Support

Reasons for seeking assistance from peer

Familiarity

[S19 in Turkish]: Arkadaşımdan öncelikle yardım almamın nedeni, arkadaşımın her gün aynı yerde olduğumuz için kendime daha çok sıcak hissediyorum.

Teacher Unavailability

[S5 in Turkish]: Hoca yardım edemeyince yapanlara baş vuruyoruz.

Reasons for Seeking Assistance from Teacher

Teacher Expertise

[S20 in Turkish]: Çünkü bu konularda daha bilgili.

Clear Explanations and Guidance

[S29 in Turkish]: Öğretmenim direkt daha güzel gösteriyordu. ... Öğretmenim anlatarak daha açıklayıcı gösteriyordu.

Self-Perceived Proficiency

[S30 in Turkish]: Ben genelde onlardan önde gittiğim için onlar geride oluyor..

Peer Support and Interaction

Constructive Collaboration

[S26 in Turkish]: Ya geliyorum, öğretmenimden izin alarak, yanına. Mesela şunu şöyle yaparsan yapabilirsin, şunu yanlış yapmışsın şu kadar açıyla yapacaktın... Öyle. Kodlarla yardım ediyordum.

[S12 in Turkish]: İstişare ederim yani arkadaşlarımla tartışarak çünkü direkt onların yaptıklarını dinlersem yine de farklı olur, anlayamam.

Unproductive Collaboration Strategies

[S21 in Turkish]: Yani beni biraz tabi ki erteliyor, sonra bakıyor benim soruma. Deniyor ilkten kendi yapmayı çözdiği soruyu. Yapamadıysa gidiyor kendi bilgisayarından açıp bana veriyor. ... Yani kendisi orada yaptığında ben tabi çok anlayamıyorum, mantık yürütemiyorum. Ama o yaptığında demek ki böyle yapmalıymışım diyebiliyorum mesela sorunun cevabını.

[S14 in Turkish]: Eğer o soruyu geçtilerse geri gelip o soruyu bana gösteriyorlardı. Pek anlayamıyordum.

Theme 4: Independent Learning Approaches

Independent Learning Strategies

Reviewing Past Solutions

[S8 in Turkish]: Eskilerine dönerim, onlara bakarım. Onlara benzeyen bir şey varsa onları geçiririm.

Trial and Error

Self-Visualization

[S16 in Turkish]: Bu gibi durumlarda kendimi hayal ediyordum. Mesela bu şeyler, kare, şunlar var ya, kareler, karelerde kendim gibi hissediyordum ve ne tarafa döneceğimi kendim belirliyordum. Atıyorum bir boydan şeklimi aynı yerde olduğumu düşünüyordum ve nereye gideceğimi yapıyordum. Ve çok kolay oluyordu.

Guidance from the Coding Platform

[S4 in Turkish]: Evet bazen böyle tam böyle kodlamaya girdiğimizde en başta video çıkıyor bir de ortalarda çıkıyor onları izliyoruz. İzleyince daha açıklayıcı oluyor aslında çıkanlar. İzlemeyince hani bunun ne olduğunu anlayamıyorsun bazen hani bir şey çıktığında.

Benefits of Solo Programming

General Positive Perceptions

Improved Focus

[S17 in Turkish]: Bilişim (Bilişim Teknolojileri ve Yazılım) dersinden daha açık olurum. Nasıl desem? Böyle sakince otururdum hocayı dinlerdim. Yanımda biri olmazdı. Dikkatim dağılıyor.

Active Engagement

[S_8 in Turkish]: Onun söylediği kodları kendim anlayamıyordum çünkü kendim bakmadan yani kodun ne olduğunu, nasıl çalıştığını anlayamıyordum. Ama kendim baktığımda daha iyi anlıyorum.

[S16 in Turkish]: Tek başına kodlama yapmanın olumlu yanı bence bütün soruları görebiliyorsun ve bütün sorulara kendin cevap veriyorsun. Yapmaya çalışıyorsun, biraz beynini çalıştırıyorsun falan bence daha iyi oluyor.

Enhanced Retention

[S14 in Turkish]: ...ama daha çok aklında kalıyordu. Daha çok soru çözüyordun çünkü.

Challenges of Solo Programming

Lack of Pair Consultation

[S22 in Turkish]: ...mesela anlamadığım bir konuyu hocaya sorduğum zaman bazenleri anlayamıyorum, ne yapacağımı bulamıyorum. Arkadaşım olduğu zaman yardım ediyordu, bulabiliyorduk beraber ama olmadığı zaman biraz zorlanıyordum.

Theme 5: Goal Setting

Mastery-Oriented Goals

Desire to Simplify Complex Tasks

[S10 in Turkish]: Akış şemalarındaki hocam şeyleri değiştirmek isterdim, görselleri. Hocam yazıyla yazmalarını isterdim. Şekillerle olmasın, yazılarla...

[S18 in Turkish]: İç-içe döngüleri değiştirmek isterdim hocam. O konuda çok kötüyüm. O konuyu çıkartmak isterdim.

Challenge Seeking

[S7 in Turkish]: Gittikçe daha zor şeyler geldi mi daha da heyecanlanıyorsun.

Career Oriented Goals

[S20 in Turkish]: Çünkü düşündüğüm ilerideki meslek yazılım mühendisliği. O yüzden önem gösteriyorum. İlğim var yazılıma o yüzden... İlğim olduğu yüzden yazılımı seçmeye çalışıyorum.

[S1 in Turkish]: Yani büyüdüğüm zaman, meslek sahibi olduğum zaman önemli olacak diye düşünüyorum. Yani şimdi büyüdüğüm zaman işlere

girdiğim zaman kodlama olur işlerde çünkü çok oluyor. Şu an pek emin değilim yani büyüdüğüm zaman olacak kesinlikle kodlama.

Daily Life Context Relevance

[S27 in Turkish]: Benim için önemli. Hayatımda zor durumlarda yardım edebilir bana. Mesela annem yemeği yapacak. Ama 'Kendin yap, ben gidiyorum' diyecek. 'Anne bana algoritma yapar mısın?' diyebilirim. 'Algoritma ne?' der. Ben de derim, ona anlatırım. O da yapar bana. Ben de kendim yaparım yani.

[S10 in Turkish]: Hocam, teknolojik aletlerde falan işimize yarayabilir. Mesela Amerika'da Tesla'ları telefonla çağırabiliyoruz. O yönden bence gerekli.

Recreational Interest in Coding

[S3 in Turkish]: Ben bunu sadece hobi olarak yapıyorum. İleride de hobi olarak yaparım sadece.

Performance-Oriented Goals

Competition Focus

[S1 in Turkish]: ...Sonra dedi ki ya biraz daha diğerlerini geçelim diye, hızlı yapalım diye ben yapayım dedi. Ben tamam dedim.

Completion-Driven Motivation

[S4 in Turkish]: Bazen biz şey yapıyoruz. Bir yanımda oturan arkadaşımın Code.org'una giriyoruz bir benim hesabıma giriyoruz. Onun hesabına girdiğimizde bende yapılmayan yerleri evde tamamlıyorum.

Academic Achievement Focus

[S24 in Turkish]: Mesela hoca sınav yaptığında hep kodlamalardan soruyor. Kodlama üzerinden soruyor. Öyle şekiller veriyor sınav kağıdında. Mesela aynı etkinlikleri yapıyoruz, hoca da aynısını soruyor, öyle. O

yüzden, o açıdan bir fayda sağlıyor bana. Bir de sınavlardan yüksek almamı sağlıyor.

Performance Avoidance Goals

Fear of Failure

[S7 in Turkish]: Çünkü etrafta çok kişi vardı. Yapamadığımda utanıyordum.

[S28 in Turkish]: Adı aklıma gelmiyor ama bazı şeyler zordu hocam. Yapamam diye korkuyordum.

Avoidance of Challenging Tasks

[S24 in Turkish]: En az sevdiğim şey zorlu kodlamalar diyebilirim yani.

Skipping Tasks

[S21 in Turkish]: Yani çoğunlukla orada üç kişiye, iki kişiye atlayalım diyoruz. Mesela yapamadığımız çok fazla örnek oldu bence. Hepsini de atladık.

Theme 6: Affective Aspects

Attitude

Positive Attitudes

Interest in Learning Programming

[S18 in Turkish]: Çok sarıyor hocam. Kodlamayı seviyorum.

Enjoyment of Plugged Activities

[S3 in Turkish]: Bilgisayar sınıfına inmek gerçekten daha iyi oldu benim için de. Eğleniyoruz. Eğitim görüyoruz bilgisayar sınıfında.

[S27 in Turkish]: Böyle yani girince yapasın geliyor. Bakınca sorulara yapasın geliyor. Çünkü hoş sorular vardı. Güzel kodlama falan vardı.

Enjoyment of Unplugged Activities

[S4 in Turkish]: Gerçekten o bardak oyununu çok sevmiştim. Bir de şey yapmıştık böyle robot şeklinde hareket etmiştik. Tahtaya yazmıştı hoca. Arkamızı dönmüştük. Bir tane arkadaşımız çıkmıştı. Birisi robot oluyordu diğeri de tahtadakileri söylüyordu. İşte dereceler oluyordu, orada kal diyordu, sağa-sola dön diyordu. O çok eğlenceli olmuştu. Güzel olmuştu.

Enjoyment of Social Interaction

[S14 in Turkish]: Ama sosyalleşerek yaptıklarımız da (bilgisayarsız etkinlikler), onlar da güzeldi. Mesela bazı derslerde okul bahçesine çıkıp orada... Bir ara bilişim hocası bizim sınıflara şey getirmişti, bir tane tavşan deliği şeyi, tavşan deliği. Tavşan havuca ulaşmaya çalışıyordu. Mesela onun gibi, onun biraz daha büyüğünü okulun bahçesinde yere resim olarak çizip oynamak.

[S26 in Turkish]: Birazcık daha sosyalleştim orada. O da benim gibi kodlamayı seviyordu. Yani bilgisayar olmasa onunla tanışamazdık çünkü.

Appeal of Familiar Characters

[S8 in Turkish]: Yani böyle bilinen oyun karakterleri falan olduğu için daha çok ilgimi çekti ve böylece yani daha kolay yaptım. Kolaylaştırdı, karakterlerin olması hem heyecan verdi hem de kolaylaştırdı.

[S22 in Turkish]: Kodlamayla ilgili, hani Angry Birds var ya, başka bir filimler de olabilirdi. Mesela Bumblebee falan, böyle robotlarla ilgili daha güzel olurdu. Daha eğlenceli olurdu, yani bir farklılık olurdu. Eğlenceli olurdu daha fazla.

Engagement of Gamification

[S3 in Turkish]: Zaten o code.org üzerinden oynadığımız oyunlar falan çok eğlenceliydi. Yani düşünürsek aslında kodlama yazıyoruz ama gerçekten çok eğlenceliydi. O 'ileri git' veyahut da oyunu bitirmek falan. Bunlar gerçekten eğlenceliydi kodlamada.

Positive Classroom Atmosphere

[S1 in Turkish]: Bir de nazik bir hoca. Nazik anlatıyor. Yani ben de mutlu oluyorum yani bağırmıyor. İlk defa görüyorum. Yani çok çok kızdırdığımız zaman bağırtıyor o kadar yani. O da direk geçiyor zaten. Hocamızı çok seviyorum. Bir şey söylediğimiz zaman o tamam diyor.

Negative Attitudes

Negative Disposition Towards Programming

[S19 in Turkish]: Sevmedim kodlamayı. Biraz anlamadım. Biraz zordu.

Frustration from Prolonged Use

[S4 in Turkish]: Mesela bir kursu bitirdiğinde diğerine geçtiğinde sıkıcı oluyor sadece. Biraz daralıyor insan. Sıkıcı oluyor gerçekten de.

[S9 in Turkish]: Başka yeni şeyler denemek isterdim.

Self-Efficacy

Confidence in Coding Abilities

[S29 in Turkish]: Ya aslında zorlanmadım ben yani. Çok kolaydı hepsi.

[S26 in Turkish]: Bir de bu işin zor olduğunu fark ettim yani. Öyle kolay bir iş değil.

Determinants of Self-Efficacy Perceptions

Social Recognition from Peers

[S26 in Turkish]: Mesela böyle giriyoruz ya kendimi profesör gibi hissediyorum nedense. Bu işlerde uzman olmuş birisi gibi hissediyorum. Arkadaşlarım bana soru soruyor, ben de 'Şöyle şöyle yaparsan yapabilirsin' diyorum. O zaman yani kendimi çok iyi hissediyorum.

Peer Comparison

[S7 in Turkish]: Başarılı görüyorum ama ... çok da iyi diyemiyorum. Çünkü benden iyi olanlar da var. Onların seviyesinde değilim. Ortalamanın biraz üstü.

Mastery Experiences

[S9 in Turkish]: Çünkü bir keresinde hoca anlatırken ben konuyu anlamıştım. Biranda böyle bitirmiştım onu. Başarılı olduğumu oradan anlamıştım, ne kadar hızlı yaptım diye.

Academic Performance

[S6 in Turkish]: Üç yazılım 100 geldi. Ondan biliyorum kodlamada başarılı olduğumu.

Perceived Cognitive Abilities

[S29 in Turkish]: Anlama kapasitem daha yüksek olduğu için...

[S1 in Turkish]: Beynim de çok almadığı için...

CURRICULUM VITAE

Surname, Name: Kefeli Berber, Pınar

EDUCATION

Degree	Institution	Year of Graduation
MS	KTÜ Computer Education and Instructional Technology	2013
BS	OMÜ Computer Education and Instructional Technology	2008
High School	Samsun Anatolian High School, Samsun	2004

WORK EXPERIENCE

Year	Place	Enrollment
2014-Present	RTEÜ Computer Technologies	Lecturer
2010-2014	Doğanköy Middle School, Trabzon	ICT Teacher
2008-2010	Karaağaçlı Middle School, Trabzon	ICT Teacher

FOREIGN LANGUAGES

English

PUBLICATIONS

1. Kefeli Berber P., Keleş E. (2022). Tele Çalışma: Kapsam ve Doğası. In Y. Karal (Ed.), *Bilgi ve İletişim Teknolojileri Aracılığıyla Uzaktan Çalışma* (pp. 1-34). Pegem A Yayınları.
2. Kefeli Berber P., Toplu N. (2022). Topics in Technology Enhanced Language Learning. In A. Çekiç (Ed.), *Topics in Technology Enhanced Language Learning* (pp. 107-130). Cumhuriyet Üniversitesi Yayınları.

3. Çelik, S., Saylan, E., Çaylak, N., & Berber, P. K. (2021). İngilizce Öğretmen Adaylarının Toplumsal Duyarlılık ve Sosyal Adalet Düzeyleri ile Yeni Medya Okuryazarlıkları Arasındaki İlişki. *Uludağ Üniversitesi Eğitim Fakültesi Dergisi*, 34(3), 1332-1372.
4. Kefeli, P. (2018, May 2-4). Ortaokul Bilişim Teknolojileri Öğretmenlerinin Kodlama Öğretimine Yönelik Görüş ve Deneyimleri [Conference presentation]. 12th International Educational Technology Conference, İzmir, Türkiye.
5. Kefeli P., İlkhan M., Güleç M., Tokel S. T. (2016). Design Report of Geography Island Project: Description of Environment and Instructional Elements [Conference presentation]. 10th International Computer & Instructional Technologies Symposium Rize, Türkiye.
6. Kefeli Berber P., Polat E., Hopcan S. (2016, May 16-18). The Perception of High School Students, Their Parents and Teachers About Essay Tests [Conference presentation]. 10th International Computer & Instructional Technologies Symposium, Rize, Türkiye.
7. Keleş, E. & Kefeli, P. (2011, May 25-27). İlköğretimde Akıllı Tahta Kullanımına Yönelik Düzenlenen Bir Hizmet İçi Eğitim Kursunun Değerlendirilmesi [Conference presentation]. 11th International Educational Technology Conference, İstanbul, Türkiye.
8. Keleş, E., & Kefeli, P. (2010). Determination of student misconceptions in “photosynthesis and respiration” unit and correcting them with the help of cai material. *Procedia-Social and Behavioral Sciences*, 2(2), 3111-3118.