Programming as a Tool for Helping Students Understand and Solve Quadratic Equations

David Taub

Faculty of Health, Science and Technology

Mathematics

LICENTIATE THESIS | Karlstad University Studies | 2024:35

Programming as a Tool for Helping Students Understand and Solve Quadratic Equations

David Taub

LICENTIATE THESIS | Karlstad University Studies | 2024:35

Programming as a Tool for Helping Students Understand and Solve Quadratic Equations

David Taub

LICENTIATE THESIS

Karlstad University Studies | 2024:35

urn:nbn:se:kau:diva-101786

ISSN 1403-8099

ISBN 978-91-7867-502-9 (print)

ISBN 978-91-7867-503-6 (pdf)

https://doi.org/10.59217/vqbe1837

© The author

Distribution:
Karlstad University
Faculty of Health, Science and Technology
Department of Mathematics and Computer Science
SE-651 88 Karlstad, Sweden
+46 54 700 10 00

Print: Universitetstryckeriet, Karlstad 2024

WWW.KAU.SE

Abstract

This study explored the idea that working on programming tasks may help students learn mathematics. The aim of the study was to determine if the act of programming itself could help students construct new mathematical knowledge. This was motivated by the idea that in order to be able to write a program to deal with general solutions and methods, the students would need to reflect on their own knowledge and make their reasoning explicit and detailed.

Three intervention cycles were conducted with different groups of Swedish secondary school students. Each intervention consisted of three separate lessons where the students worked on completing programming tasks. Good data was obtained from one group of two lower secondary school students (competent programmers) and two groups of two to three upper secondary school students (novice programmers). The interventions themselves were modeled on task-based interviews though the overarching methodology and analysis was based on the Abstraction in Context theoretical framework. The results were also analyzed to find evidence of proceptual thinking as well as creative mathematical reasoning.

The findings indicate that although it seems possible for the act of programming itself to help students construct new mathematical concepts, it is unlikely to be an effective classroom strategy. The cognitive difficulties the novice programmers encountered when writing programs became an obstacle to learning new mathematics as well as dramatically increasing the time needed to complete apparently simple tasks. The findings also indicate that even competent programmers may not be able to effectively learn mathematics through programming as they may find programming solutions to tasks that bypass the desired mathematical content.

Table of Contents

Preface 1
1 Overview 3
1.1 What This Thesis Is About 3
1.2 What This Thesis Is Not 5
1.3 A Few Key Phrases and Formatting Notes 6
2 Introduction 7
3 Related Research 10
3.1 Research on Programming and Mathematics 10
3.2 Research on Learning to Solve Quadratic Equations 15
4 Research Questions 18
5 Context: Programming in Swedish Schools 19
5.1 Changes Made in 2018 for Grades 1-9 20
5.2 Changes Made in 2018 for Upper Secondary School 21
5.3 Updates Made in 2022 for Upper Secondary School 22
6 Background 23
6.1 Computational Thinking 23
6.1.1 What Is Computational Thinking? 23
6.1.2 Computational Thinking and Programming 25
6.1.3 Critiques of Computational Thinking 27
6.1.4 Abstraction and Computational Thinking 28
6.2 Learning to Program 29
6.2.1 Programming Is Hard to Learn 30
6.2.2 But Is It Actually Hard to Learn? 31
6.2.3 What Do Students Have Trouble With? 32
6.2.4 Comparing Programming Languages 35
6.2.5 Syntax Errors 38
6.2.6 Variables 41
6.2.7 Loops 48
6.2.8 Conditional Statements 51
6.2.9 Structure and Planning 51
6.2.10 What Can Be Done to Help Students Learn Programming? 52
6.2.11 A Note on Tinkering 53

6.3 Quadratic Equations 54
6.3.1 What Is a Quadratic Equation? 54
6.3.2 Solving a Quadratic Equation 55
6.3.3 More on Factoring and the Zero Product Property 58
6.3.4 A Brief History of Quadratic Equations 62
7 Theoretical Background 74
7.1 Proceptual Thinking 74
7.2 Creative Mathematical Reasoning 75
7.3 Competing Views on Learning 77
7.4 What Is Abstraction? 78
7.5 Algebraic Thinking 80
8 Abstraction in Context: A Theory and a Methodology 82
8.1 The Frist Step 83
8.2 RBC+C Model 83
8.3 Shared Knowledge 85
9 Method 86
9.1 Task-based interviews 87
9.2 A Priori Analysis 88
9.3 Task Design 88
9.4 Ethical Considerations 94
9.4.1 Personal Data 95
9.4.2 The Ethical Review Act 95
9.4.3 Information and Consent 95
9.4.4 Anonymity 96
10 How to Read the AiC Analyses 97
10.1 Additional Theoretical Considerations 98
11 Intervention 1 99
11.1 Data Collection in Intervention 1 99
11.2 A Priori Analysis for Intervention 1 100
11.3 AiC Analysis of Intervention 1 104
11.3.1 C_0: The Zero Product Property 106
11.3.2 C_{fi}: Factors of an Integer 107
11.3.3 C_e: Understanding Equations and Expressions 108

11.3.4 C_{2s}: Two Solutions to a Quadratic Equation 110
11.3.5 C_{mb}: Multiplying Two Binomials 112
11.3.6 C_{ffs}: Connection Between Factored Form and Solutions..... 113
11.3.7 C_{st}: Connection Between Standard Form and Solutions..... 115
11.4 Summary of Results From Intervention 1 118
12 Intervention 2..... 121
12.1 A priori Analysis for Intervention 2..... 122
12.2 Data Collection in Intervention 2..... 122
12.3 AiC Analysis of Intervention 2 123
13 Intervention 3..... 124
13.1 A Priori Analysis for Intervention 3 124
13.2 Data Collection in Intervention 3..... 125
13.3 AiC Analysis of Intervention 3 Group 1..... 125
13.3.1 C_{fi}: Factors of an Integer 128
13.3.2 C_{e}: Understanding Equations and Expressions..... 130
13.3.3 C_{mb}: Multiplying Two Binomials 131
13.3.4 C_{ffs}: Connection Between Factored Form and Solutions..... 131
13.3.5 C_{st}: Connection Between Standard Form and Solutions..... 134
13.4 AiC Analysis of Intervention 3 Group 2..... 138
13.4.1 C_{0}: The Zero Product Property..... 139
13.4.2 C_{fi}: Factors of an Integer 140
13.4.3 C_{mb}: Multiplying Two Binomials 140
13.4.4 C_{ffs}: Connection Between Factored Form and Solutions..... 141
13.4.5 C_{st}: Connection Between Standard Form and Solutions..... 146
13.5 Summary of Results From Intervention 3 150
14 The Effects of Previous Programming Experience..... 152
14.1 Student Difficulties Stemming from Lack of Experience 152
14.1.1 Difficulties With Variables and the Notional Machine..... 153
14.1.2 Difficulties With Loops..... 154
14.1.3 Difficulties With Syntax..... 155
14.2 Student Difficulties Stemming from Previous Experience 156
15 Discussions 157
15.1 Method Discussion..... 157

15.1.1 Knowledge Construct Granularity 158
15.1.2 Methodology Problems 159
15.1.3 Practical and Ethical Problems 159
15.2 Results Discussion 161
15.2.1 Findings in Support of the Aim 161
15.2.2 Findings in Support of RQ 1 163
15.2.3 Findings in Support of RQ 2 164
15.2.4 Findings not in Support of the Aim, RQ 1 or RQ 2 165
15.3 Future Research 166
15.4 Final Thoughts 167
References 171
Appendix 1 Programming Tasks 185
A1.1 Tasks Used in Intervention 1 185
A1.1.1 Task Overview 186
A1.1.2 Task 1: Two Constrained Numbers Whose Product Is 0 187
A1.1.3 Task 2: Generate Equations for the Previous Task 188
A1.1.4 Task 3: A Quadratic Equation From Two Solutions 189
A1.1.5 Task 4: Find All Signed Factor Pairs of an Integer 190
A1.1.6 Task 5: Factor a Quadratic Equation With Integer Solutions 191
A1.1.7 Task 6: Solve a Quadratic Equation With Integer Solutions 192
A1.2 Tasks Used in Intervention 2 193
A1.2.1 Changes Made to Tasks After Intervention 1 193
A1.2.2 Task Structure 194
A1.2.3 When Should the Teacher Help the Students? 195
A1.2.4 The Output Should Include Calculation Details 196
A1.2.5 Task Overview 197
A1.2.6 Task 1: Find All Signed Factor Pairs of an Integer 199
A1.2.7 Task 2: Output the Sum of the Factor Pairs 201
A1.2.8 Task 3: Output Two Binomials 202
A1.2.9 Task 4: Output the Product of Binomials 204
A1.2.10 Task 5: Evaluating an Expression 206
A1.2.11 Task 6: Find the Solutions to an Equation in Factored Form 208
A1.2.12 Task 7: Produce a Quadratic Equation From Its Solutions 210

A1.2.13 Task 8: Factor a Quadratic Equation With Integer Solutions 212
A1.2.14 Task 9: Solve a Quadratic Equation With Integer Solutions 216
A1.3 Tasks Used Intervention 3 217
A1.3.1 Task Overview 218
A1.3.2 Programming Preparation Tasks 219
A1.3.3 The Main Tasks 221
A1.3.4 Task 1: Factoring an Integer 221
A1.3.5 Task 2: Find the Sum of the Factors 222
A1.3.6 Task 3: Evaluating a Quadratic Equation in Factored Form 222
A1.3.7 Task 4: Solving a Quadratic Equation in Factored Form 223
A1.3.8 Task 5: Changing From Factored Form to Standard Form 224
A1.3.9 Task 6: Generating a Quadratic Equation from Two Solutions 225
A1.3.10 Task 7: Factoring a Quadratic Equation in Standard Form 226
A1.3.11 Task 8: Solving a Quadratic Equation in Standard Form 227
A1.3.12 Final Task on Paper: Applying the Lessons 228
A1.3.13 Example Programs and Functions 228
Appendix 2 Coded Transcription Lines 232
A2.1 Coded Lines From Intervention 1 232
A2.1.1 C_0: The Zero Product Property 232
A2.1.2 C_e: Understanding Equations and Expressions 234
A2.1.3 C_{2s}: Two Solutions to a Quadratic Equation 235
A2.1.4 C_{mb}: Multiplying Two Binomials 237
A2.1.5 C_{fi}: Factors of an Integer 238
A2.1.6 C_{ffs}: Connection Between Factored Form and Solutions 239
A2.1.7 C_{st}: Connection Between Standard Form and Solutions 245
A2.2 Coded Lines from Intervention 3 Group 1 250
A2.2.1 C_{fi}: Factors of an Integer 250
A2.2.2 C_e: Understanding What an Equation Is 252
A2.2.3 C_{mb}: Multiplying Two Binomials 252
A2.2.4 C_{ffs}: Connection Between Factored Form and Solutions 253
A2.2.5 C_{st}: Connection Between Standard Form and Solutions 259
A2.3 Coded Lines from Intervention 3 Group 2 265
A2.3.1 C_0: The Zero Product Property 265

A2.3.2 C_{fi}: Factors of an Integer 266
A2.3.3 C_{mb}: Multiplying Two Binomials 267
A2.3.4 C_{ffs}: Connection Between Factored Form and Solutions..... 268
A2.3.5 C_{st}: Connection Between Standard Form and Solutions..... 276

Preface

I imagine every thesis is its own personal journey, with highs and lows and everything in between. I thought it might be interesting to write a little bit about how mine began, as well as some idea of why this text might be of interest to various readers.

Even though most of my official degrees say "mathematics" on them, I have always had a strong personal interest in programming. I grew up when computers first started appearing in homes, and like many motivated teenagers at the time, I taught myself to program.

I have always been interested in the intersection between computers and other fields of study. My first degree was in neurobiology, which, combined with my interest in computers, led me naturally into studying neural networks.

The final motivation for this project came from the recent international push to teach more programming to more students, in particular the inclusion of programming lessons in mathematics classrooms in Sweden. If mathematics teachers have to teach programming as part of their mathematics lessons, it seemed natural to ask how programming could be used to teach mathematics, a question which formed the basis of my thesis.

There are several aspects of this text that may be of interest to a range of readers. Most obviously, mathematics teachers in Sweden may find information in this text of interest since it is directly relevant to how they are required to exercise their profession today.

Mathematics teachers in general may also find the background on quadratic equations interesting and useful. This includes a detailed history of how humans learned to solve quadratic equations as well as a discussion of common problems students have when learning to solve them in a classroom (as reported in the current literature).

Programming teachers in other countries may also find useful information here, whether or not they teach it as part of mathematics. This text contains a fairly detailed overview of problems students have when learning to program, and the references can lead to even more relevant information for teaching the novice programmer.

Other researchers may find inspiration for new projects or interesting questions to ask.

And finally, those who are politically interested, especially in Sweden, may find good source material here to use when discussing and reflecting on the changes to the school curriculum connected to programming.

1 Overview

1.1 What This Thesis Is About

This thesis explores the use of programming as a method of learning mathematics. The aim of this study was to examine whether or not the act of programming itself can help students develop a deeper understanding of mathematics, as opposed to focusing on the results or output of a program. The study looked at whether the specific cognitive requirements to complete a series of carefully chosen programming tasks would encourage students to form a deeper understanding of the mathematics being focused on in the tasks. To this end, small groups of students in three separate interventions were given a series of programming tasks to work on during three different lessons. The tasks were designed to help the students learn how to solve quadratic equations by factoring.

The choice to focus on quadratic equations began by looking at other research combining programming with mathematics. Most of this research seemed to take advantage of a computer's ability to help visualize mathematics, leaving obvious gaps in research looking at aspects of mathematics without an obvious visual focus. This limitation in the literature seems to have been enhanced by a bias towards the use of visual programming environments (see section 3.1 for more information).

Algebra seemed like a natural choice to consider here, leaving the question of which aspect of algebra to focus on. Narrowing the choice down to quadratics equations was mainly motivated by two additional factors. First of all, there is a lack of research on problems students encounter when learning to solve quadratic equations, indicating a need for more research in this area in general (see section 3.2 for more information). Secondly, quadratic equation are first taught in upper secondary school, which would theoretically have given the students time to have already learned some basic programming in lower secondary school before participating in the intervention.

Data was collected during the interventions using a task-based interview methodology (Maher & Sigley, 2020) based on a cyclical design process, where the results of each intervention were used to

improve the tasks and intervention process. I took on the role of both teacher and researcher since the ordinary teacher was not present for any of the interventions. See section 9.1 for more information.

Audio recordings of the students' discussions, as well as screen recordings and all written material, were collected and analyzed using the Abstraction in Context theoretical framework (Dreyfus & Kidron, 2014). This detailed qualitative theory and methodology is used to categorize student utterances in order to find evidence for the construction of new mathematical knowledge. This methodology has been used to good effect by other researchers looking at how programming can help students learn mathematics. For example, Anabousy et al. (2023) used Abstraction in Context to analyze whether or not programming with Scratch could help Israeli 6th graders learn about coordinate systems. See section 8 for a more detailed explanation of Abstraction in Context.

This study's first intervention took place at a lower secondary school with two 9th grade students who had extensive experience programming in C++. Both students were high academic achievers, especially in mathematics, and had volunteered for the project.

The second intervention was at an upper secondary school with two groups of first year students. A series of difficulties resulted in no useful data being obtained from either group.

The third and final intervention was at a different upper secondary school with two groups of students, one group of two and one group of three, all in their first year. All five of these students were high academic achiever's and had volunteered for the project. Only one of these students had any previous programming experience.

All of the interventions were at schools in Sweden, though the first school had an English profile. The students spoke almost entirely in Swedish, though the students in the first intervention mixed in the occasional bit of English. The transcriptions in the appendix have all been translated into English (my translations).

1.2 What This Thesis Is Not

In addition to explaining what my project is about, it seems equally important to explain what it is not. In particular, my choice of project is not to be taken as an endorsement, explicitly or implicitly, of the current system of mathematics education in this, or any other, country.

After first contrasting the inherent tensions between the needs of society and the needs of the individuals within that society, Niss (1994) described the following three large-scale problems with mathematics education:

- *The justification problem*. Why should we teach the mathematics we teach? This question has also been discussed by Ernest (2019): "mathematics in education needs a justification, and one that is critically balanced and does not prejudge the outcome because of conventional historical assumptions or the views of insiders or ideologically slanted interest groups" (p. 3).
- *The possibility problem*. Is it possible to teach students the mathematics as stipulated by the relevant governing bodies?
- *The implementation problem*. What is the most effective way to teach the required mathematics to the required groups of students?

My project will mainly deal with the implementation problem, though it will touch on the possibility problem as well. Although I consider it to be of great importance, my project will not deal with the justification problem at all. Instead, like Lundin (2017), I will view mathematics education as a required ritual in our society and my job will be to find ways to help students attain the rewards offered to those who can successfully complete the ritual.

It also seems important to be clear that I am not making any claims about, or even look at, whether or not programming is the best or most efficient method for teaching any part of mathematics. This thesis doesn't even broach the question of whether or not it is a good method. Instead, this is an attempt at a "proof of concept", looking only at the possibility, and perhaps finding some conditions and outlines for when that possibility can be achieved. Ideas of efficiency and "goodness" compared to other methods are left for future research if that seems warranted. The territory needs to be identified before it can be mapped.

1.3 A Few Key Phrases and Formatting Notes

Table 1 below lists definitions of some important programming terms as well as some notes on how fonts are used for clarity.

Table 1

Definitions of Programming Terms

Phrase	Explanation
programming	This text will use the definition of programming found in Kelleher and Pausch (2005, p. 83): "the act of assembling a set of symbols representing computational actions. Using these symbols, users can express their intentions to the computer and, given a set of symbols, a user who understands the symbols can predict the behavior of the computer." This includes both block-based programming and text-based programming.
block programming block coding visual programming environment	These phrases all refer to the use of programming languages and environments such as Scratch and Snap! where the user creates a program by dragging and dropping pre-created visual code elements, referred to as "blocks". These blocks snap together like virtual Legos to build a program.
text-based programming	This phrase refers to the use of programming languages such as C, C++, Java and Python where the user writes the program using a text editor.
programming block	Not to be confused with block programing, a programming block is a section of code in a text-based programming language that is connected in some way. For example, this may refer to the contents of a loop or the code to execute after a conditional statement has been met.
Scratch	The most popular block-based programming environment. Designed specifically to help younger children learn to program.

Programming code appearing in text will be shown in the font `Courier New` for clarity.

Note that all references to research involving programming in this text should be assumed to refer to text-based programming unless otherwise stated.

2 Introduction

Over 40 years ago, Papert (1980) suggested that computers have the potential to revolutionize how students learn mathematics in school. At the time, he complained that research was missing this potential and thus limiting the opportunity for progress in how students could learn mathematics. Papert felt that most of the new ideas being presented were stuck within a limited paradigm and thus were not so new after all. To help elucidate this idea, he made the analogy to how constant improvements on a horse-drawn carriage will never produce an automobile: an entirely new paradigm is required. He envisioned computers enabling students to live in what he called *Mathland*, a land where they would learn the language of mathematics as naturally as a child growing up in Sweden learns Swedish. This would be accomplished by a series of well-constructed programming tasks, commonly referred to as microworlds, using a suitable programming environment and language (Logo and turtle graphics in his examples).

Over a decade later, Noss and Hoyles (1996) expressed their own view on a need for radical changes to the way mathematics is taught and learned, framing most changes at that point as “pedagogy without epistemology” (p. 4). Like Papert (1980), they thought these changes could be realized through the use of computers. However, they make the point that the idea of transferring knowledge from one domain to another (e.g., from computer mathematics to pen and paper mathematics) was already being called into question in the 1990’s. They thought the focus instead should be on using the computer as a tool for expressing mathematics since “setting is intimately bound up with performance on mathematical tasks” (p. 30).

Although neither Papert's (1980) nor Noss and Hoyles' (1996) dreams of immersion in computer programming as a way of learning mathematics has come to pass, research has continued in this area, and programming is increasingly being combined with mathematics lessons in various ways around the world.

This thesis explores taking Papert’s idea a step further. Instead of relying on programming tasks related to mathematics automatically creating a need to develop mathematical ideas, the tasks were designed to try and specifically encourage this. The goal was to focus students'

attention on details in the mathematics and encourage them to make ideas connected to these details concrete in a way that, perhaps, pen and paper work may not encourage. The focus here was on the act of programming itself and not so much on the finished program or its output. Good results using a similar approach were seen by Freudenthal et al. (2013) when they created tiny programs that "focus student attention on exploring principles underlying (and building 'gut level' intuitions related to) the content of high-school level algebra and pre-calculus courses" (p. 3).

According to Hoyles and Noss (1992), programming can act as a bridge between the general and the specific. The authors go on to suggest that programming can act as a scaffold to allow students to explore ideas in mathematics by “debugging their own intuitions through a process of inductive generalisation” (p. 452). However, Hoyles and Noss (1992) emphasize that task design is a key element if programming is to help students learn mathematics.

In addition, the ability of a computer to carry out repeated calculations and test a large number of cases was taken advantage of to allow students to focus on developing and understanding underlying concepts instead of just focusing on the procedures, and thus hopefully develop their *proceptual thinking* (see section 7.1 for more detailed information on proceptual thinking). Similar ideas were explored by Tall (1992), who saw positive results from using programming to help students understand algebra expressions. In Tall's study, the students used programming to do the "heavy lifting" with the calculations, allowing the students to focus on a more conceptual understanding of the expressions.

There is another theoretical framework which lends support to the idea that the act of programming can help students deepen their understanding of mathematics. After discussing the literature on different types of reasoning, Lithner (2008) provides a clear framework for disentangling creative mathematical reasoning from imitative reasoning. In this framework, imitative reasoning does not lead to any deeper mathematical understanding and a student engaging in it is rarely even learning or practicing the intended concepts, while creative mathematical reasoning does often lead to

deeper mathematical insights. See section 7.2 for a more detailed discussion of Lithner's theoretical framework.

Creating programming tasks which the students don't have previous knowledge of an overall algorithm or procedure for will put them in a position where they will be encouraged to engage in creative mathematical reasoning in order to complete the task.

As can be seen above, there is ample reason to believe that there may be a way for programming to be beneficial to the process of learning mathematics. However, some researchers have identified potential problems and risks when attempting this. Here are a few of the more pertinent ones.

- If the programming tasks are too hard, the students may lose interest in both the mathematics and the programming (Dohn, 2020). An attempt to avoid this problem was made in this study by constructing appropriately leveled tasks based on the students' expected previous knowledge.
- There is a risk the students will get lost in the technical details of programming and lose focus on the mathematics (Borg, 2021; Ke, 2014). To try and avoid this problem in this study, the programming requirements of the tasks were kept as simple as possible. In addition, example code and technical programming help was provided to students to help them with aspects of programming that weren't directly related to the mathematics being learned.

Unfortunately, the later problem proved to be quite an obstacle in the second and third interventions, an issue which will be discussed in greater depth later in this text.

3 Related Research

This section explores recent research directly connected to the main focus of this thesis: combining programming and mathematics and problems students encounter when learning to solve quadratics equations, especially when using the zero product property (see section 6.3 for information on quadratic equations and methods for solving them).

3.1 Research on Programming and Mathematics

Most of the recent research into programming and mathematics appears to focus on similar programming environments as well as a limited number of topics within mathematics. The block programming environment Scratch is overrepresented in the literature, as are aspects of mathematics that lend themselves to visual solutions or simple visualizations. This limitation is not only due to the inherently visual nature of Scratch itself since using programming as a method for visualizing mathematics is common in research based on other programming languages as well (e.g., Karavakou & Kynigos, 2020; Lambić, 2011).

A reoccurring theme seen in the literature is using Scratch to introduce programming to younger students, both with the goal of teaching computational thinking (see section 6.1 for a detailed discussion of computational thinking) as well as teaching concepts within mathematics. For example, Brennan and Resnick (2012) developed a framework to help them understand and assess how the use of design-based activities, such as programming with Scratch, promote the development of computational thinking in young students. In England, the ScratchMaths project was designed to introduce 9-11 year-olds to computational thinking specifically to help them explore mathematics, with most of the topics connected to geometry (Benton et al., 2016). Also in England, Benton et al. (2017) reported on using programming in Scratch to help middle school students visualize and understand a 360 degree turn. Middle school students in USA were a bit more positive towards mathematics after using Scratch to create mathematical games (Ke, 2014). Also looking at the effects of computational thinking on mathematics, Calao et al. (2015) saw an increase in skills in mathematics among Columbian sixth graders after

three months of training with Scratch. In contrast to some of these results, Dohn (2020) saw a reduction in interest in both programming and mathematics after Danish sixth graders learned to use Scratch in the *Programming in Maths* curriculum, which also had a heavy focus on topics connected to geometry. This loss of interest was attributed to the tasks being too hard and too boring.

Aside from topics connected to geometry, using programming to help students with problem solving also appears in the literature. This includes visual based problem solving as well as other methods, such as trial and error. In Sweden, Borg et al. (2020) observed secondary school students trying to use programming to solve mathematical problems mainly using trial and error. Calao et al. (2015) looked at how mathematics skills improved in Columbian sixth graders after learning to program with Scratch for three months and found a very slight increase in the students' problem solving abilities. Also focusing on problem solving, Psycharis and Kallia (2017) failed to see any positive effects in Greek secondary school students from learning to program. In Spain, Rodríguez-Martínez et al. (2020) saw only modest improvements in computational thinking and problem solving abilities among sixth graders who worked on programming tasks in Scratch.

Other areas of mathematics do appear in the literature, even if less frequently. For example, middle school students in Turkey and in Cyprus showed an increase in their understanding of probability after programming games with Scratch (Akpinar & Aslan, 2015; Paparistodemou et al., 2017). American fourth graders showed an improvement in mathematical skills after writing games to teach fractions to even younger children (Kafai & Burke, 2015). On top of this, there has been some research into using programming in the teaching of university level mathematics (e.g., Buteau et al., 2019; Cline et al., 2020; Karavakou & Kynigos, 2020; Lovric, 2018; Sangwin & O'Toole, 2017).

A few researchers have looked at whether students "automatically" create mathematical meaning through the act of programming. For example, Ke (2014) had American students design computer games in Scratch to teach ideas in mathematics. In this study, the authors didn't just look at the mathematics in the game itself, but also at whether or

not the students learned additional mathematics from the act of programming itself connected to the Scratch programming environment. While the process of programming seemed to confer some increase in the understanding of coordinate systems and the idea of variable, as well as a slightly more positive view of mathematics, the students often got lost in the details of coding. In addition, the mathematics in the game itself was often just a simple numerical calculation thrown on top of a game shell.

A literature review of programming in mathematics education for students between the ages of six and sixteen found no strong evidence that programming improved student understanding in mathematics (Forsström & Kaufmann, 2018). In a recent case study of the use programming in mathematics in Swedish classrooms in grades 2, 6 and 9, Stigberg and Stigberg (2020) reported that mathematics teachers questioned the introduction of programming into mathematics classrooms and didn't report seeing any change in mathematics skills resulting from their use of programming. It is worth noting that these examples include uses of programming that didn't use tasks designed to increase the understanding of a specific concept within mathematics, which suggests that the programming tasks need to be well thought out and planned if they are to be effective in increasing understanding within mathematics (Noss & Hoyles, 1996; Papert, 1980).

Research into how programming can help students understand algebra has been rather limited so far (note that it is beyond the scope of this text to discuss the important, though complex, question of what, exactly, algebra is). Most of the connections to algebra that have been looked at are related to built-in features of the chosen programming environment, such as coordinate systems and variables. For example, American sixth graders showed an improvement in their understanding of coordinates after working with Scratch (Germia & Panorkou, 2020). In their description of the British ScratchMaths project, Clark-Wilson et al. (2019) explained that multiple uses of the `Answer` block in Scratch was intended to motivate middle school students to learn how to use variables in order to save previous answers. They also claimed that this motivated teaching programming in mathematics classrooms. In Denmark, Misfeldt and Ejsing-Duun

(2015) noted that fifth graders learned about variables and coordinates in order to make the games they were programming more fun.

When discussing the connection between learning to program and understanding variables, it is important to note that there is a degree of ambiguity when looking at the concept of variable. The common English use of the word *variable* includes what Ely and Adams (2012) divide up into *unknowns*, *variables*, *generalized numbers* and *placeholders*. It is not obvious that understanding the use of variables in computer programming will help students understand all the other ways letters are used in algebra, something students are known to struggle with (Bardini et al., 2005; MacGregor & Stacey, 1997). Analyzing the semantics and syntax of programming languages, Bråting and Kilhamn (2020) point out that variables are used differently in programming than in algebra, and that this may add to students' confusion when using letters in mathematics instead of helping them. Looking at students learning to program in BASIC, Bayman and Mayer (1983) found that some students had trouble understanding the difference between assigning a value to a variable (the common programming use of the equal sign as an assignment operator) and storing an equation in memory (viewing the equal sign as it is commonly used in mathematics). This problem was also seen more recently by Kohn (2017) while teaching Python to advanced placement students in upper secondary school. In their study, some students thought that the variable assignment `g=x*x` stored the equation `x*x` in memory instead of its current value. The students assumed that if the value of x was changed later then the value of g would automatically be updated according to the saved equation. Kohn (2017) called this type of misunderstanding *deferred evaluation*. See section 6.1 for a more detailed treatment of common programming misconceptions.

A small amount of additional research can be found looking at how programming can facilitate students' meaning-making in algebra. Bråting and Kilhamn (2020) performed a semiotic analysis of similarities and differences between programming and algebra, highlighting areas where programming may help, as well as others where it may interfere with students' understanding. In the end, they were not able to come to any definitive conclusions. Bråting et al.

(2020) describe a project to look at how programming in Swedish mathematics classrooms affects the learning of algebra. The project focused on analyzing teaching materials and teacher thoughts, opinions and experiences, but did not collect direct empirical data on student learning. Freudenthal et al. (2013) looked at the implementation of a teaching module called “iMPaCT-Math” which was intended to use programming to help American 9th graders learn Algebra I, a secondary school mathematics course in the American school system. The study looked at the classroom implementations of the learning module after a group of teachers were trained in how to use it, but their initial results ran into problems when, even after the training, the teachers lacked the requisite knowledge to implement the module. For example, teachers who overemphasized procedural fluency in their day-to-day teaching would provide too much help which interfered with the students' ability to spend time on analysis. A similar problem with inadequate teacher knowledge was seen by Holmes et al. (2018) who looked at training Australian teachers to implement the ScratchMaths project in their classrooms, although they did report an improvement in the teachers' ability to integrate coding and mathematics after the training.

Schanzer (2015) looked at the feasibility of an intervention called “Bootstrap” in which American eighth and ninth graders programmed a game using a functional programming language called “Racket”. He found evidence of transfer from Bootstrap to the traditional pen and paper domain, but the results were significantly mediated by the individual teachers conducting the interventions more than the students themselves. Several years later, an even larger study was done by Schanzer et al. (2018) where they found that after completing the Bootstrap module, students improved their ability to solve algebra word problems (e.g., asking the student to write a function that represents a phone bill related to the number of minutes spent talking on the phone given some information about the pricing plan).

Some researchers have identified additional problems that may arise for students trying to combine programming and mathematics. For example, while studying upper secondary school students using programming to solve mathematical problems, Borg (2021, p. 172) noted that the students had difficulty transferring "their mathematical

ideas into programming code" and that "students tried to use mathematical notations within their program in a way not compatible with the way a code is executed". It is worth noting that Borg (2021) attributes these problems to a combination of the students' inexperience with programming as well as to their difficulty understanding how programming could help them solve the problems they were assigned.

3.2 Research on Learning to Solve Quadratic Equations

Although there has been a good deal of research into common problems students encounter when solving and trying to understand first-degree equations, there has been relatively little comparable research involving quadratic equations. An examination of the existing literature on the topic shows that a number of authors have identified difficulties with quadratic equations which overlap with those already known for first-degree equations. For example, some students were unable to solve quadratic equations because of difficulties working with fractions (O'Connor & Norton, 2016), difficulties working with negative numbers (O'Connor & Norton, 2016; Yahya & Shahrill, 2015), inadequate basic computational skills (Kotsopoulos, 2007; Yahya & Shahrill, 2015), difficulties with basic algebraic manipulation (Tall et al., 2014), as well as exhibiting various misunderstandings connected to the concept of variable (Didiş Kabar, 2018; Didiş et al., 2011; Vaiyavutjamai & Clements, 2006).

However, a number of problems specific to solving quadratic equations have also been identified:

- Misunderstanding the structure of quadratic equations, including confusing them with first-degree equations (Makonye & Nhlanhla, 2014).
- Difficulties with applying the quadratic formula and related formulas such as the pq-formula used in Sweden (Didiş & Erbas, 2015; López et al., 2016; Olteanu & Holmqvist, 2012).
- Difficulties with completing the square (Didiş & Erbas, 2015; López et al., 2016).

- Poor understanding of square roots, in particular not knowing that there is both a positive and a negative solution to equations such as $x^2 = 9$ (López et al., 2016; Tall et al., 2014).
- Difficulties with factoring and understanding the zero product property (Didiş et al., 2011; Didiş & Erbas, 2015; López et al., 2016; Memnun et al., 2015; Vaiyavutjamai & Clements, 2006).

Errors connected to factoring and understanding the zero product property include those seen by Adler and Davis (2006) who report 10th graders in South Africa trying to solve $x^2 - 2x = -1$ by factoring the left side directly and getting $x(x - 2) = -1$. The same problem was also pointed out by Didiş et al. (2011) where tenth graders in Turkey tried to solve $x^2 - 14x + 24 = 3$ by factoring the left side and setting the factors equal to 3. Similar difficulties could also be seen elsewhere in the literature (e.g., Kotsopoulos, 2007; Makonye & Nhlanhla, 2014).

Other misunderstandings about the zero product property were also observed, in particular problems which were thought to be connected to a misunderstanding of the concept of variable. This was seen when students checked the solutions to a factored quadratic equation by putting in both solutions at the same time, treating each x like a different variable and ending up with $0 \cdot 0 = 0$ (e.g., Didiş et al., 2011; Hu et al., 2021; Vaiyavutjamai & Clements, 2006). For example, when checking the solutions to $(x - 3)(x - 2) = 0$, they substituted $x = 3$ **and** $x = 2$ back in at the same time, thinking that x could simultaneously have two different values and that the x in each factor was in essence a different variable, instead of understanding that $x = 3$ **or** $x = 2$ and getting either $0 \cdot 1 = 0$ or $-1 \cdot 0 = 0$.

Some students, and even some teachers, when presented with a factored quadratic equation, such as $(x - 3)(x - 2) = 0$, first multiplied out the left hand side and then either refactored it or tried to use the quadratic formula, showing a poor understanding of the zero product property and perhaps an overly procedural understanding of how to solve a quadratic equation (e.g., Hu et al., 2021; Makonye & Nhlanhla, 2014; O'Connor & Norton, 2016).

Other factoring mistakes were reported by Olteanu and Holmqvist (2012), where Swedish students tried factoring $x^2 + 6x + 5 = 0$ and got

$x(x + 6) + 5 = 0$ and then decided that this meant the solutions were $x = 0$ and $x = -6$.

Additional conceptual errors were reported by López et al. (2016), where students divided both sides of $(x - 4)(x + 4) = (x - 4)$ by $(x - 4)$ thus losing a solution, and by Didiş et al. (2011), where students tried to solve $x^2 - 2x = 0$ by first moving the $2x$ over to the other side of the equal sign and then dividing both sides by x, also losing a solution.

Many, if not all, of these errors have been attributed to students likely having a purely procedural understanding of factoring and the zero product property.

4 Research Questions

The aim of this project is to add to the growing body of knowledge currently exploring the possibilities of using programming to facilitate the learning and understanding of mathematics. It is of interest to explore the extent to which programming can enhance the teaching and learning of mathematics, and in doing so, to identify which aspects of programming are most beneficial as well as identifying any guiding principles for designing programming tasks which can be used to help students learn mathematics.

Current research in this area continues to be generally focused on narrow uses of programming as well as limited areas of mathematics. Notably absent from the current literature is an exploration of programming tasks designed specifically to help students make meaning of techniques for solving quadratic equations. Therefore, this project will attempt to answer the following research questions:

- In what ways can programming tasks facilitate student learning and understanding of how to use the zero product property to solve quadratic equations?
- Which aspects of the programming tasks are helpful in facilitating the avoidance of common misunderstandings previously identified in the literature which are associated with using the zero product property to solve quadratic equations?

5 Context: Programming in Swedish Schools

A large part of the motivation for this study were the changes made to the Swedish curriculum introducing programming as requirement in mathematics courses. This section provides an overview of these changes.

The full history of the introduction of computers and programming into Swedish schools is long and complex and has been covered in depth by a number of authors (e.g., Raptopoulou, 2021; Vinnervik, 2022). This text will focus on the two most recent, and probably largest, changes: the requirement to teach programming from elementary school through upper secondary school as part of other subjects, including mathematics, in 2018 (Skolverket, 2018a), and the updates to these requirements in 2022 (Skolverket, 2022a).

In 2018, the Swedish curriculum for elementary school through upper secondary school were amended to include the teaching of programming as a part of mathematics (Skolverket, 2018a). Before these changes were introduced, programming was, and continues to be, offered as its own subject in upper secondary school as an elective. Teachers for these programming courses are expected to have a specific university education in the teaching of programming, something that mathematics teachers in Sweden do not generally have.

Although some researchers have been positive, at least theoretically, to the idea of teaching computational thinking as part of other subjects (e.g., Weintrop et al., 2016), there has been a fair share of criticism when looking at the actual implementation, especially, as in the Swedish case, when this mainly means teaching programming in mathematics.

In the final report of a three-year project to help teachers integrate programming in mathematics, Jahnke (2020) claimed that programming showed no benefits for helping students learn mathematics. On top of that, most of the teachers interviewed could not see any clear way to use programming to teach mathematics and instead ended up teaching programming as its own subject even if it was considered to be within mathematics lessons. A number of other researchers found similar issues with mathematics teachers having

difficulty seeing the connection between programming and mathematics (e.g., Bråting & Kilhamn, 2022).

In addition, teaching programming in mathematics may introduce extra confusion, since, as pointed out by Olteanu (2022, p. 2050), programming tasks intended for teaching mathematics differs from tasks intended for teaching programming in that the former tasks are "already formulated and accessible to the students" as well as being "intended to allow students to discern and make sense of key aspects of both mathematics and programming as well as to reason about those aspects to arrive at a solution". This is similar to the problems found by Bråting and Kilhamn (2022) when they looked at how programming was integrated into Swedish mathematics textbooks. They found evidence that including programming in mathematics may make it harder to learn programming since the programming tasks in the textbooks tend to focus om mathematical concepts, such as patterns, instead of the relevant programming education concept, such as condition.

5.1 Changes Made in 2018 for Grades 1-9

In 2018, the Swedish curriculum for grades 1-9, LRG11, was revised to include programming in mathematics (Skolverket, 2018a). Programming is first mentioned as a general requirement under the section describing the purpose of mathematics where it states that students should be given the opportunity to learn how to use digital tools and programming to explore mathematical problems, mathematical concepts, perform calculations and to present and interpret data. Note that programming is mentioned together with the more general term *digital tools* in such a way that a reader could be led to believe that the author of that part of the curriculum thinks that learning to program is on par with learning to use a piece of finished software.

For grades 1-9, programming is included in mathematics as a core content of algebra, which some researchers find surprising (e.g., Bråting & Kilhamn, 2022) since previous research on algebraic thinking has not drawn any connections to programming or even computational thinking. While Kilhamn et al. (2022) suggested the possibility of programming being able to help students understand

some concepts in algebra such as variable, they found that this was not the case in practice.

Programming is then mentioned more specifically in the contents for each of the school divisions in Sweden: grades 1-3, grades 4-6 and grades 7-9.

In grades 1-3, the only reference to programming appears under algebra, and it suggests that the students learn how to construct and follow step-by-step instructions as a basis for learning programming later. It does go on to state that this instruction should include symbolic representations for the instructions. Note that there is no explicit reference to needing to do any actual programming on a computer here.

In grades 4-6, programming appears under algebra again, and the requirement is increased to require actual programming. However, the requirement is limited to visual programming environments such as Scratch.

In grades 7-9, programming appears in two places, under algebra and under problem solving. Under algebra, the requirement is the same as for grades 4-6, except "visual programming environments" is replaced with "different programming environments", strongly implying a requirement for text-based programming. Under problem solving, the requirement is to be able to program algorithms to solve problems.

5.2 Changes Made in 2018 for Upper Secondary School

Programming was also added in 2018 to some of the mathematics courses in upper secondary school in Sweden (Skolverket, 2018b). Upper secondary school in Sweden is divided into three main tracks, with the lower-level math courses being slightly differentiated between the tracks.

- The a-track is for students pursuing a vocational education. These students rarely intend to continue on to university. The mathematics courses in this track are Mathematics 1a and 2a.

- The b-track is for students pursuing a more theoretical education outside of STEM fields. This track prepares the students for continuing on to university, even if not all of them do so. The mathematics courses in this track are Mathematics 1b, 2b and 3b.
- The c-track is for students pursuing a STEM education. A high percent of these students will continue on to university studies. The mathematics courses in this track are Mathematics 1c, 2c, 3c, 4 and 5.

In 2018, programming was added to problem solving in the courses Mathematics 1c, 2c, 3b and 3c, and was not added at all to the other mathematics courses. In all of these courses, teaching was required to cover "strategies for solving mathematical problems, including the modelling of different situations, with and without digital tools and programming" (my translation).

5.3 Updates Made in 2022 for Upper Secondary School

Larger changes were made to Swedish upper secondary school mathematics courses in 2022, including the requirements for programming (Skolverket, 2022b). Programming was added to mathematics 4 and 5, and the text was changed for the other courses.

The programming requirements for Mathematics 1c and 2c are identical, with the teaching needing to cover "examples of how programming can be used as a tool for problem solving, data handling or implementation of numerical methods" (my translation). The only difference in the text for the other mathematics courses is that the phrase "examples of" was replaced with "the use of", implying that a higher degree of competency should be taught.

6 Background

This section contains a deep dive into background information relevant to this thesis, including, a thorough analysis of computational thinking, a discussion of known issues involved with learning to program and a full explanation of quadratic equations and their solutions, including a historical overview of how those solutions were developed.

6.1 Computational Thinking

Since the desire to teach *computational thinking* has been one of the driving forces in introducing computer science, including programming, into school curriculums around the world (Mannila et al., 2014), it seems necessary to discuss computational thinking to provide context and background for this text. However, due to some of the questions and problems connected to computational thinking (discussed below), as well as its lack of relevance to the specific ideas being explored in this research project, it will not play a prominent role in analyzing data obtained in this study.

6.1.1 What Is Computational Thinking?

The first use of the term *computational thinking* is generally attributed to Papert (1996), though he only mentions it once and it is not entirely clear what he means by it. He relates the idea of computation to a new way of thinking about concepts such as gravity. His idea is to think of gravity as an object with properties which can be used in computations, what he calls a *computational object*. However, computational thinking seems to be less important to his point here than the idea of what he refers to as "object-oriented theory" (Papert, 1996, p. 110).

A perhaps more relevant first use of the term computational thinking is attributed to Wing (2006). In her heavily cited seminal article, Wing (2006) attributes a vast array of properties to computational thinking without ever really pinning down what it means. This initial attempt at a definition is so vast and all-encompassing that it would be difficult to make use of it. She fixes this issue in a later article where she provides the following concise definition: "Computational Thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be

effectively carried out by an information-processing agent" Wing (2010, p. 1).

Despite the more concise definition, it is clear from this second article that Wing (2010) still attributes an impressive degree of power to computational thinking, which she refers to as "the new literacy of the 21st Century" (p. 3).

Wing's articles sparked international interest, motivating more and more researchers to look at and try to define computational thinking. This led to a proliferation of definitions and explanations within a few years, which Grover and Pea (2013) tried to sort out in a review of research connecting computational thinking and education in the USA. They motivated their article by referring to a number of reports claiming there was a shortage of computer scientists combined with a growing job market, as well as there being too few women in the field and unequal access to basic computer science education around the country.

According to Grover and Pea (2013), attempts to define computational thinking at this point could be divided into two broad groups: theoretical definitions and defining computational thinking as a set of practices. In addition to competing definitions, the authors also found confusion regarding how to assess computational thinking, a problem highlighted several years later by Tedre and Denning (2016).

The problem persisted, with Denning (2019) calling computational thinking a buzzword that was still producing competing definitions. The author went on to try and provide his own definition of computational thinking: "Designing computations that get computers to do jobs for us, and explaining and interpreting the world as a complex of information processes" (p. 4). Here Denning (2019) clarifies that he does not consider doing a job to be the same as automation. He also emphasizes that computers are essential to computational thinking, which means that the technical details of the computer, such as memory and CPU capacity, have a profound effect on what problems can actually be solved with that particular computer. As he explains it, the bulk of computational thinking is getting a computer to do something for us. Note that this is a substantial step

away from earlier, in particular Wing's, broad claims about computational thinking being applicable to almost all areas of life.

Going in a different direction, Kallia et al. (2021) suggest that there are advantages to defining computational thinking broadly and flexibly. The authors examined existing research connecting computational thinking and mathematics and found that empirical papers looking at the implementation of computational thinking in mathematics classrooms mainly focus on automation, abstraction, modeling, algorithmic thinking, visualization, decomposition and pattern recognition. The majority of papers focusing on practical implementations highlight visualization, mostly through programming, as the most natural intersection between computational thinking and mathematics. In contrast, there is not much connected to data analysis, testing or debugging. Kallia et al. (2021) were able to identify three key aspects for teaching computational thinking within mathematics: problem solving, cognitive processes and transposition (the rephrasing of a problem so that it can be solved or explored with a computer).

Testing their ideas in a case study, Hansen and Hadjerrouit (2022) used programming as a way of connecting computational thinking and mathematics. The authors suggest students can use computational thinking to complement mathematical thinking to translate a mathematical problem or task into a solution that can be implemented through programming.

After acknowledging the problem with competing definitions of computational thinking, Vinnervik and Bungum (2022) highlight the general trend of using programming to implement it. The authors avoid adding yet another definition to the mix by using the framework developed by Weintrop et al. (2016) to analyze the implementation of computational thinking in schools in Finland, Norway and Sweden.

6.1.2 Computational Thinking and Programming

A recurrent theme in the literature around computational thinking is its connection to programming. This raises the important question of what, exactly, this connection is. Most of the literature does not address this issue directly, and so the authors' views often need to be inferred.

Looking at existing research, Grover and Pea (2013) claim that programming is an essential part of computational thinking and refer to an international push to make programming a more commonplace skill and more accessible to a larger part of the population. The authors also found that most teachers at the time were teaching and assessing computational thinking through programming, even if some of them claimed to be assessing other skills. On top of that, in some countries like England, the inclusion of programming in the curriculum was in direct response to pressure from the IT industry suffering from a lack of programmers (Tamborg, 2022). Given this, it seems important to untangle whether or not programming is indeed the primary skill needing to be assessed, or simply a pathway to other skills and knowledge.

Some authors, such Weintrop et al. (2016), have looked at the connection between computational thinking and programming. Though Weintrop et al. (2016) don't state it explicitly, the way they describe real-world applications of computational thinking seems to imply a belief in a high degree of overlap with programming. Despite this, they present a framework for assessing computational thinking practices that is not dependent on programming alone.

Other examples of researchers who implicitly equate computational thinking and programming can be found in Gomes et al. (2022), who examined the effects of the European MindMaths project in Portugal, and in Liu et al. (2023), who looked at the effects of programming games on student motivation in Shanghai.

Researchers have also found similar views among teachers. For example, Niemelä (2017) showed that Finish teachers equated learning computational thinking with learning to program.

Some attempts have been made to separate the two ideas, including Mannila et al. (2014), who discuss the difficulties with using programming as the main method of developing computational thinking competencies. The authors suggest there is value in other approaches if computational thinking really is the main goal. Even Denning (2019), who claims that computers are essential to computational thinking, and that the main application of computational thinking in the workplace is through the programming

of simulations to model different aspects of our world, emphasizes that computational thinking is not just a set of concepts for programming.

Some countries, such as Finland, Norway and Sweden, have included programming explicitly in their school curriculum without ever mentioning computational thinking, even if the idea of computational thinking was theoretically the driving force behind the changes (Vinnervik & Bungum, 2022). This could imply a national view equating learning to program with learning computational thinking skills. Note that in Norwegian "computational thinking" has been translated to "*algoritmisk tankegang*", which literally translates back into English as "algorithmic thinking". This may of course effect how computational thinking is viewed in Norway.

Some researchers have approached the issue from the other direction. For example, Kandemir et al. (2020) make the claim that teaching computational thinking is important when trying to teach programming.

6.1.3 Critiques of Computational Thinking

The rampant international popularity of computational thinking has not been devoid of criticism. As Denning (2017) points out, there are a host of problems stemming from the ever growing number of competing definitions. To start with, computational models are not included in most of the operational definitions of computational thinking. According to Denning (2017), this is a mistake, since the other aspects that are usually included, such as abstractions and decomposition, are just steps necessary to get a computer to do something useful. Furthermore, Tedre and Denning (2016) claim that implementations that only focus on programming are missing other potentially important aspects of computational thinking, which has led to voices criticizing the international focus on computational thinking as a way of smuggling in programming under another name.

The lack of clarity concerning whether computational thinking is a knowledge domain or a skill has led to implementation and assessment problems (Tedre & Denning, 2016). Skills need time and practice to master, and should be assessed by demonstrating the skill itself. However, computational thinking is usually assessed as knowledge and

not as a skill. Mastering knowledge does not automatically confer a related skill. For example, someone may know the physics of juggling without actually being able to juggle. The reverse is also true; someone may be able to juggle without knowing anything about physics. Similarly, using digital tools, either casually or professionally, does not automatically lead to computational thinking (Denning, 2017).

According to Tedre and Denning (2016), there may be even deeper problems with computational thinking. Claims of computational thinking's broad scope and positive effects on cognitive abilities have been exaggerated since the 1960's when similar claims were made of its precursor, algorithmic thinking. This includes claims that teaching computational thinking leads to better problem solving skills in other areas. As Denning (2017) also points out, there is no evidence that computational thinking transfers to any other tasks, in particular "everyday tasks", nor that it enhances any other cognitive abilities, including problem solving. This is problematic since it is precisely these broad claims which have motivated the inclusion of computational thinking in schools around the world.

6.1.4 Abstraction and Computational Thinking

This section will provide a brief overview of the role of abstraction in computational thinking and how it differs from the ideas of abstraction that will be used in this project (see section 7.4).

In their Delphi study, Kallia et al. (2021, p. 171) asked the following three questions:

1. What characterizes computational thinking in mathematics education?
2. What are the common aspects of computational and mathematical thinking?
3. Which aspects of computational thinking can be addressed in mathematics instruction?

Although the word "abstraction" appears in the answers to all three questions, as well as appearing in the summary of their results, the term is never clearly defined in their text. A similar problem can be seen in the systematic text analysis done by Kalelioglu (2016), where the word "abstraction" appears prominently in the word cloud they created

for computational thinking. However, it is possible to see hints of an implied view of abstraction in their article when the authors refer to the definition of computational thinking given by the International Society for Technology in Education (ISTE) and the Computer Science Teacher Association (CSTA), which includes the line "abstractions such as models and simulations" (p. 585). This, together with other ways abstraction is used in the article, indicates a view of abstraction as a decontextualization.

This view of abstraction in computational thinking, which is directly connected to Jean Piaget, can also be seen in Niemelä (2017) and is exemplified by "pattern generalizations, symbol systems and representations" (p. 71). The author also claims that aspects of programming, such as data types and data structures, are inherently abstract, which indicates a view that abstraction is a property of an object or construct itself and not of a person's relationship to that object or construct.

Abstraction is also mentioned, but not defined, in Weintrop et al. (2016), where the authors create a taxonomy of computational thinking for use in mathematics education. However, even here, the authors use the term in a way that implicitly indicates a view of abstraction as a decontextualization. For example, the authors use subroutines and data structures as examples of abstractions, as well as stating that in order to create an abstraction, a student needs to be able to “conceptualize and then represent an idea or a process in more general terms” (p. 139).

As can be seen, the idea of abstraction as a decontextualization seems to be the dominant view within theories connected to computational thinking, which differs from the relational view that will be used in this thesis (see section 7.4).

6.2 Learning to Program

As this study will focus on using programming to help students learn mathematics, it seems important to discuss the issues and difficulties specifically related to learning how to program. It is worth noting that this type of discussion is surprisingly absent from other discussions

and articles debating the merits of adding programming as a requirement in other subjects such as mathematics.

One of the issues here may be that a search through existing literature quickly reveals that the majority of relevant research is on first year university students taking an introduction to programming course, a problem previously highlighted by Grover and Pea (2013). Fortunately, the tide has begun to turn, with the last few years showing a rapid increase in research connected to students learning to program as early as elementary school.

Another problem negatively influencing research into programming education is the lack of researchers and especially professors who work mainly in the field of computer science education (Blikstein, 2018). Part of the problem seems to be that research into pure computer science is considered more valuable and more prestigious and so it has an easier time getting funding.

6.2.1 Programming Is Hard to Learn

It is not uncommon to hear that programming is particularly hard to learn, in particular text-based programming. This idea is so persuasive that it has motivated a number of studies, including two from the same team looking at actual failure rates in introductory programming courses (Bennedsen & Caspersen, 2007, 2019). The authors start the first paper with the sentence "It is generally accepted that it is very difficult to learn programming" (p. 32), and then go on to refer to other authors who also make the same claim.

Perkins et al. (1986, p. 41) claim that "a number of pitfalls in the process of programming can interfere with their making the best use of the problem-solving abilities to bootstrap themselves toward a reasonable level of competence." Here the authors are implying that inherent difficulties with programming are causing problems for students. They go on to say that some students just stop when they are stuck, not knowing what to do and having no idea of how to proceed; they just give up.

Reviewing the literature on problems students encounter when learning to program, Robins et al. (2003) also make the claim that learning to program is not easy, a sentiment that can still be seen in

more recent papers, such as the one by Bosse and Gerosa (2017) with the title "Why is programming so hard to learn?". This idea is repeated a year later, though in a somewhat more nuanced paper. In their systematic review of problems with introductory programming courses, Luxton-Reilly et al. (2018) discuss other research indicating that programming is hard.

However, there may be good reason to question this persistent belief.

6.2.2 But Is It Actually Hard to Learn?

When Bennedsen and Caspersen (2007) looked at actual failure rates in introductory programming courses at universities around the world, they found that around 33% of the students failed the course. Repeating their study 12 years later, the authors found the fail rate had dropped to 28% (Bennedsen & Caspersen, 2019). In both papers, the authors argue that there are good reasons to think that these failure rates are not particularly high, especially when compared to the 50% fail rate for first year university algebra in the USA.

In another international study, Watson and Li (2014) found similar pass and fail rates. The authors also saw no significant change in pass rates since the 1980's, as well as no difference in pass rates between programming languages being learned. Interestingly, they found higher pass rates for high school students compared to university students, and they also saw considerably higher pass rates for smaller classes.

Taking a harder stance, Luxton-Reilly (2016) claims that not only is programming not inherently hard, but saying so is unfair to students and can impact issues such as diversity and equity by reducing opportunities for some groups to gain access to programming skills. After reviewing the literature, the author came to the conclusion that "students are not able to program at the level expected by instructors by the end of an introductory programming course" (p. 286). In other words, it's not that programming is inherently difficult, but rather that expectations are too high and the tasks being assigned are too difficult. This sentiment was repeated in a later review by the same author on specific difficulties students have when learning to program (Luxton-Reilly et al., 2018).

Other research finding positive effects from changing how programming is taught adds weight to the idea that programming is not inherently difficult and that the problem may instead be with how it is taught. For example, Fujiwara et al. (2012) found that well-timed advice helped Japanese university students make progress when stuck. In Turkey, Cetin et al. (2014) saw positive results by having university students answer discussion questions in conjunction with their programming tasks to help them reflect on what they were doing and how they were thinking. Also in Turkey, Öztürk et al. (2021) saw positive effects from teaching block-based programming to 6th graders using project based learning instead of a more traditional style of teaching individual programming concepts. Reviewing the literature on teaching programming through gamification, Kučak and Kučak (2022) saw the potential for gamification as a good method for teaching programming to younger novice programmers. However, the authors point out that there is still a lot of room for improvement here.

At this point, it seems fair to say that learning to program, though not easy, is perhaps not actually as inherently difficult as some might suggest. That being said, there is a fair bit of research looking at the specific aspects of programming that students struggle with.

6.2.3 What Do Students Have Trouble With?

A review of the existing literature on programming education turns up a number of specific concepts and structures that students often struggle to understand. However, before proceeding with a discussion of these concepts and structures, it is important to tackle what appears to be a major underlying conceptual problem. A repeated theme in much of the literature is that many students have trouble understanding what is often referred to as the *notional machine*, which is an understanding of how a computer works and how it runs a program (du Boulay, 1986). In other words, what is really going on "behind the curtain" when a computer runs a program. Reviewing the literature on problems with learning how to program, Robins et al. (2003) found that problems with understanding the notional machine continued to be an important source of mistakes, a problem that seems to persist today (Qian & Lehman, 2018).

As Denning (2019, p. 9) puts it "Computers are incredibly dumb. They perform mindless, mechanical steps extremely fast but they have no understanding of what the steps mean." This may seem obvious to someone more familiar with computers and programming, but it is apparently more difficult to fully understand for novice programmers.

The essence of the problem seems to be that students expect computers to do what they mean and not what they say (du Boulay, 1986). According to Pea (1986), this seems to stem from students making an unconscious analogy that programming a computer is like talking to a human. Calling student misunderstandings *bugs*, Pea (1986) categorized the bugs he found into different groups. However, many of the examples in the different groups can still be connected to problems with the notional machine. For example, with his *parallelism bugs*, students thought different lines of code would be active at the same time. Consider the following bit of Python code which demonstrates this issue.

```
x=2
if x==3:
    print("Hello")
x=3
```

The students in question would think this code would print "Hello" because the variable `x` was changed to 3, even though it was after the `if` statement. This can be seen as the students thinking of the above `if` statement as a general instruction to the computer, similar to what you might say to a person: "Print *hello* whenever x is equal to 3." The author's *intentionality bugs* were more directly stated as the students assuming the computer could somehow know their true desires and intentions. In his *egocentrism bugs*, students expressed openly that they thought the computer should do what they mean and not what they say.

Two other papers published in the same year looked at upper secondary school students learning Pascal and Basic (Putnam et al., 1986; Sleeman et al., 1986). Students in both groups went so far as to think that the computer knew what they wanted based on the name of the variables alone. For example, they were asked to write a program that asked the user to input several different numbers which would then be

stored in different variables. In both groups, some students thought that the computer would automatically store the largest number in the variable named "largest", regardless of what the code actually looked like. Furthermore, some of the students learning Pascal thought that a print statement asking the user for input would work the same as the input command. For example, they thought that the following line alone would cause the computer to ask the user for input:

```
print("Input a number")
```

This problem seems to be both persistent and international. Looking at university students learning to program in Malaysia in 2009, Tan et al. (2009) found that the students were having trouble learning to program because they didn't understand how the computer actually ran the code they were writing. In their review of the literature on programming misconceptions, Qian and Lehman (2018) saw that one of the problems was that students did not always understand that the code was run sequentially. For example, students thought that the control conditional in a `while` loop would trigger whenever it became true, not just once per loop. While this could be seen as a problem specific to loops, it can also be seen as a problem with the notional machine.

It seems clear from the literature that any serious attempt at helping students learn to program should start with helping them understand the notional machine. That being said, there is plenty of research on other specific difficulties students encounter when learning to program. Some of these difficulties are sometimes referred as misconceptions. This text will use the definition of a misconception provided by Chiodini et al. (2021, p. 381): "A programming language misconception is a statement that can be disproved by reasoning entirely based on the syntax and/or semantics of a programming language."

As the above definition indicates, many misconceptions are specific to a given programming language, and either don't exist or are in fact true in other languages, which means they cannot always be said to be wrong in some absolute sense (Chiodini et al., 2021). This indicates that comparing programming languages would be a good starting point.

6.2.4 Comparing Programming Languages

Today there are a great variety of programming languages to choose from, with different strengths and weaknesses depending on their intended use. Although some of the earliest languages have evolved into more modern versions, such as Visual Basic from the original Basic, many of them, such as FORTRAN and Pascal, are not seen as much today. Some modern languages, such as C#, are designed for real-world use, while others, such as the block programming environment Scratch, are designed for educational purposes. While simplified programming languages may help novice programmers learn to program, there is also a risk that poorly thought out simplifications may instill misconceptions that are hard to overcome when progressing to "real" programming languages (Blikstein, 2018).

Another important distinction is between compiled and interpreted languages. With compiled languages, such as Java, C, C++ and C#, the programmer must run a compiler on the program to turn it into a binary code that the computer can execute directly. Interpreted languages, such as Python, do not need to be compiled and are instead executed by another already compiled program which interprets the program directly. Compiled programs generally run faster than interpreted programs, though advances in technology have closed the gap a bit. A compiler can also be instructed to optimize the code in a number of ways. For example, if memory is the primary limiting resource, the compiler can be instructed to compile the code in such a way as to save memory. If execution speed is more important, this can be prioritized instead. Note that optimizing speed often uses a lot more memory. For example, the compiler may "unroll" loops in the source code to have fewer commands to execute (this removes the jump back instruction at the end of each loop), or "inline" functions (this removes function calls). These can add up to a significant savings in speed, but at the cost of using more memory.

Although compiled languages are important for most high-performance applications, they have a few disadvantages. A notable disadvantage is that a compiled executable can only run on one particular type of computer. This means that a C++ program compiled

for Windows will not run on Mac OS, and vice versa. In general, this isn't problem for scripted programming languages.

The literature is rife with arguments between researchers about which languages are better for teaching programming, with some researchers going so far as to create their own language after dismissing all available choices. In a review of existing literature on novice programmers, Luxton-Reilly et al. (2018) found no evidence for different languages leading to different learning outcomes. However, a few years later Lokkila et al. (2023) found that Python seemed to be a better choice than at least Java when looking at error rates among novice programmers.

There are a lot of factors to consider when choosing which language to teach to novice programmers, including trends and changes in programming paradigms. For example, most modern languages tend to be object oriented, especially for real-world applications (Lokkila et al., 2023), but Sim and Lau (2022) found evidence that it was easier to learn imperative programming first instead of object oriented programming. Depending on the age being taught and the goal with the course, ease of learning may need to be weighed against real-world application.

There has been a growing trend to develop new languages and development environments that are easier for novice programmers to learn (Kelleher & Pausch, 2005). Changes include making syntax easier and developing better tools to help learners detect and fix errors. It seems reasonable to make the case that this is a valid line of approach when introducing programming for the first time to younger students, in comparison to university students in a computer science program planning on working in the field.

To help highlight the differences in syntax between different programming languages, it may be of interest to look at a very simple program written in several different programming languages. The examples below, each of which calculates the sum of cubes up to an integer specified by the user, are presented in chronological order based one when each language was created. When run, all four programs do the exact same thing and produce the same output.

Basic

```
DIM sum as integer
DIM i as integer
DIM number as integer
INPUT "Enter a positive integer: ", number
sum = 0
FOR i = 1 TO number
  sum = sum + i ^ 3
NEXT i
PRINT "The sum of cubes to "; number; " = "; sum
```

Pascal

```
program sum_of_cubes;
Uses Math;
var
  number, i, sum: integer;
begin
  write('Enter a positive integer: ');
  readln(number);
  sum := 0;
  for i := 1 to number do
  begin
    sum := sum + trunc(power(i, 3));
  end;
  writeln('The sum of cubes to ', number, ' = ', sum);
end.
```

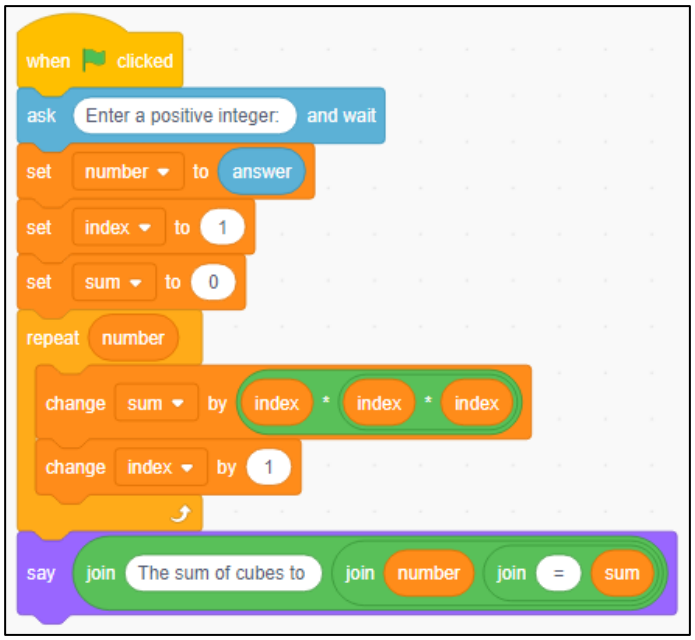
C++

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
    int number, i, sum = 0;
    cout << "Enter a positive integer: ";
    cin >> number;

    for (i = 1; i <= number; i++) {
        sum += pow(i, 3);
    }
    cout << "The sum of cubes to " << number << " = " << sum
<< endl;
    return 0;
}
```

Python

```
number=int(input("Enter a positive integer: "))
sum=0
for i in range(number+1):
    sum = sum + i**3
print(f"The sum of cubes to {number} = {sum}")
```

Scratch



As can be seen, there are considerable differences in the amount of overhead needed to create a simple program, whether or not variables need to be declared, how loops are delineated, what functionality is included in the core language, how exponentiation is handled, and even in the symbols used for variable assignment, not to mention how input and output is dealt with. So even if the underlying programming logic is similar between languages, language-dependent syntax clearly plays an important role.

6.2.5 Syntax Errors

Syntax differences between languages can be big and, not surprisingly, seem to be one of the contributing factors to making some languages easier to learn (Lokkila et al., 2023). Mistakes with syntax are commonly referred to as *syntax errors*. Interestingly, none of the literature examined for this text defines *syntax error*, even when listing them. Most seem to implicitly adhere to the definition in the Oxford dictionary of computer science, which this text will also use: "A programming error in which the grammatical rules of the language are broken" (Oxford, 2016).

Syntax errors were identified early on as one of the most common types of programming errors (e.g., du Boulay, 1986; Pea, 1986; Putnam et al., 1986; Sleeman et al., 1986), a problem which continues to appear in the

literature. The problem is so big that it has been driving the development of new programming languages with easier syntax (Kelleher & Pausch, 2005). In some ways, Scratch, and other block programming languages, are the ultimate result of these attempts since it is not possible to make a syntax error in these languages by their very nature.

A common type of syntax error is connected to the equal sign. In most programming languages, there is potential confusion between the way equal signs are used in mathematics and in programming, a problem which is exacerbated by the difference between assignment and comparison when programming (Bråting & Kilhamn, 2020). To make matters worse, different programming languages use different symbols for these operators, though most of the more modern languages seem to be settling on a similar pattern. Table 2 below shows a few examples.

Table 2

Assignment and Comparison Operators in Different Programming Languages

Language	Assignment	Comparison
Basic	=	=
Pascal	:=	=
C, C++, C#, Java, Python	=	==

Another important syntax differences is how blocks of code are delineated; in the languages based on C (e.g., C++ and Java), coding blocks, such as those under an `if` statement or within a `for` loop, are enclosed within braces {}, whereas Python uses different amounts of indentation. It is not immediately clear which method is better; Lokkila et al. (2023) suggests that while indentation may feel more natural, it is more sensitive to errors and may inadvertently encourage problems with the notional machine where students act as if the computer knows what they mean. To help add to the confusion, languages that use braces generally allow their omission if the block only contains one statement. For example, compare the following snippets of code in C#.

One statement in the block: braces not required.

```
if (rhino == 1)
    elephant = 3;
```

More than one statement in the block: braces required.

```
if (rhino == 1) {
    elephant = 3;
    zebra = 4;
}
```

To make matters worse, a program block is only considered one statement, so does not require braces if it is nested inside another block and is the only statement. This applies to multiple nesting layers as well. In addition, white space is ignored and so indentation is only for human readability. This means that it is syntactically correct to do the following in C#:

```
if (a==1)
for (int i=0; i<5; i++)
if (i==3)
for (int j=0; j<5; j++)
if (j>2)
c=3;
```

The way each programming statement needs to be ended can also differ between languages. In Python, there is no particular symbol needed, just a normal return. However, if you need to break a long statement in Python into multiple lines, you need to end the partial statement with a backslash "\" to show that the statement actually continues. In most C languages, as well as in Pascal, statements need to end with a semicolon ";". In JavaScript, the semicolon is optional. A number of researchers have found that forgetting the semicolon is one of the most common syntax errors (e.g., du Boulay, 1986; Garner et al., 2005; Lokkila et al., 2023). The problem with syntax errors is international, with Bosse and Gerosa (2017) seeing the same issue among university students in Brazil and So and Kim (2018) seeing it with fourth through sixth graders learning Python in Korea.

Looking specifically at syntax errors with novice programmers, Denny et al. (2011) suggested that syntax errors may interfere with exercises where the intention is to focus on problem solving or developing an algorithm. Focusing on university students learning Java in New

Zealand, the authors found that even the top students were unable to write syntactically correct code more than half the time, whereas the problem was so great for the weakest students that they were unable to get almost any code to compile at all, which prevented them from getting useful feedback from the teachers. Note that this is a problem specific to programming languages that need to be compiled. The authors also saw that students tended to try and fix one syntax error at a time and then recompile, which is not a particularly efficient work process.

In their review of the literature on programming misconceptions, Qian and Lehman (2018) found that syntax errors were still the most common error, however they also claim that they were often the easiest to fix. The authors list the most common syntax errors as being mismatched brackets and quotes, missing semicolons and mixing up assignment and comparison operators.

Given their prevalence, it is clear that syntax errors have a significant impact on how easy it is for a student to learn how to program. However, it can be difficult to know where to draw the line between syntax errors and other types of errors. For example, when students mix up assignment and comparison operators, it is not always clear if it is a simple matter of forgetting the correct syntax or if it is because the students do not understand the difference between the two operators. This raises the question of whether or not this is just a syntax error, an error with variables, an error with conditionals, or if it belongs in some other category altogether.

As a last note on syntax errors, it is worth pointing out something not seen in the literature. Most programming languages are based on English: in Sweden, students use `if` and `for` instead of the Swedish `om` and `för`. It seems reasonable to think that this may cause an added layer of difficulty for students who don't have English as their first language, but it is hard to find any mention of this in the literature, so perhaps it is not much of an issue after all.

6.2.6 Variables

As already seen, problems with variables can overlap with syntax errors, misunderstanding the notional machine and more. However, it

is an important enough and a big enough problem area to deserve its own section. In order to draw an arbitrary line somewhere, as well to keep the focus on problems more likely to be seen with novice programmers, more advanced issues such as those with arrays, pointers and function parameters will not be discussed here.

The literature is rife with problems novice programmers have using and understanding variables. This difficulty has led to a variety of methods for trying to explain variables to students, perhaps culminating in Byckling and Sajaniemi (2006) who introduced the idea of variable roles while teaching Pascal to Finish university students. The authors came up with ten different roles which variables take on in 99% of all programming projects. Although their results were not conclusive, with all of their students having trouble learning programming, they do claim to have seen some improvement in the students using the roles in some situations.

A more traditional explanation is to say that a variable is like a box. The programmer can then put a number in that box and later look at it or change it. Unfortunately, this explanation runs into problems from the start; we can store other objects in variables, such as strings and Boolean values, or even references to more complicated objects. It's not even simple limiting the discussion to just numbers, since the numbers themselves can be signed or unsigned integers, single or double precision floating point numbers, or single bytes. And these are just some examples.

The box analogy can even fail on the simplest level. For example, du Boulay (1986) saw students misusing the analogy to think that a variable can hold more than one value at the same time, a problem also seen by Sleeman et al. (1986), Putnam et al. (1986) as well as Qian and Lehman (2018) in their literature review.

A less common error where students did essentially the opposite of this was described by Pea (1986) who saw students using the same variable for different values at the same time. One of the examples given was when the students were asked to write a program to calculate the average of some numbers input by the user, they used the same variable for the input and to hold the running total.

Here is an example of what this looks like in Python:

```
for i in range(5):
    x = int(intput("Enter a number: "))
    x = x + x
average = x/5
```

6.2.6.1 Scope

A common problem students have with variables is understanding scope, which can be seen in the literature at least as far back as du Boulay (1986). Scope appears as number 18 in a comprehensive list created by Garner et al. (2005) of problems encountered by novice programmers, it is mentioned by teachers in Brazil as a major problem among students (Bosse & Gerosa, 2017), and is discussed by Qian and Lehman (2018) in their literature review.

In programming, a variable's scope is the part of the program where it is both defined and has a consistent value. In other words, the scope of a variable defines where in the program that particular variable name must be unique in order to avoid conflict. A variable which can be used everywhere in a program is called a *global* variable. Variables which aren't global are *local* variables and have a more limited scope. In case this seems like it might not be confusing enough, the rules for the scope of a variable can be different in different programming languages.

In Python, a variable defined outside of a function is generally global and one defined inside a function is generally local. However, even in Python it can be confusing.

This example will print "global" twice. Both occurrences of the variable `x` being printed are the same.

```
x = "global"
def checkVariable():
    print(x)
checkVariable()
print(x)
```

This example will print "local" and then "global". When the variable `x` is assigned inside the function, a local version of the variable is created disconnected from the global version with the same name.

```
x = "global"
def checkVariable():
    x="local"
    print(x)
checkVariable()
print(x)
```

In languages like C#, variables can be even more local than just inside or outside of a function. The following code snippet will cause an error when compiled since the variable `x` is only defined inside the `for` loop and so the program is trying to print a variable that doesn't exist after the loop.

```
for (int i=0; i<3; i++){
    int x;
    x=i;
}
x=3;
```

There is much more that can be said here, but this gives the basic idea of what is meant by scope and why it might be confusing.

6.2.6.2 Data Types

Students have also been seen to have difficulty understanding and managing data types when working with variables, which appears as error number 20 in Garner et al. (2005). In many programming languages, a variable can only hold one type of data, which is determined when the variable is first created by the program. The most common data types are signed and unsigned integers, floating point decimals numbers, Boolean values and strings.

Like most of the topics being discussed here, there is quite a lot that can be said about data types. However, to keep within the limited scope of this text, only the most basic points will be highlighted. One of the most important differences between programming languages is whether or not variables have to be declared before being used. Consider the following two examples.

In Python, variables can be used without being declared.

```
x=3
```

In languages based on C, variables have to be declared before they can be used, though an initial value can be assigned in the declaration step.

```
int x=3;
...
x=4;
```

Declaring variables has several functions. Perhaps most importantly, it acts as a safety check on the programmer to prevent them from accidentally assigning the wrong type of data to the variable at some point in their program. Some programmers will even intentionally put extra constraints on their variables to help avoid mistakes in complex programs. What this means in practice is that if a variable is declared as an integer, that is the only type of data it can be assigned and trying to assign something else, like a floating point number, will cause the compiler to issue a warning.

This C# code will produce an error when compiled.

```
int num;
num=3.5;
```

Note that even in languages where variables need to be declared, it is possible to change incompatible data before it is stored in a variable by "casting" the data to the correct type. However, in some cases this will result in losing information.

This Java example won't produce any errors. However, `num` will equal 3 at the end since the process of casting 3.9 to an integer involves removing everything after the decimal point.

```
int num;
num = (int)3.9;
```

Python, on the other hand, is dynamically typed. This means that variables do not need to be declared in Python, and that their type will be automatically changed based on the value being assigned to them.

This Python code will not produce any errors. The variable will simply change types automatically.

```
num = 3
num = 3.9
num = "Now I'm a string"
```

There are advantages and disadvantages with both methods for novice programmers as can be seen in the literature. For example, Bosse and

Gerosa (2017) found that students had trouble understanding the intersection of arithmetic and data types. In particular, they didn't always understand that when dividing two integer types, some languages will cast the result to an integer thus truncating any decimals. This is less of an issue in dynamically typed languages like Python.

The following is an example of the errors seen by Bosse and Gerosa (2017) in Java. This snippet will print 0 and not 0.5 as might be expected.

```
int num = 1;
int den = 2;
System.out.println(num/den);
```

This Python snippet will print 0.5 as expected.

```
num = 1
den = 2
print(num/den)
```

Comparing Java and Python, Lokkila et al. (2023) claim the dynamic data typing in Python should be easier for novice programmers, an idea supported by So and Kim (2018) who did not find any data type errors among young students learning Python in Korea. Trying to present a balanced view, Lokkila et al. (2023) mention the possibility that dynamic typing may make some errors harder to find. Other advantages of strongly typed languages include code which is easier to read and easier to maintain as well providing a host of optimization benefits for compilers. However, most of these advantages are not usually relevant for novice programmers.

6.2.6.3 Assigning Values to a Variable

Trouble with variable assignments is also common in the literature (e.g., Putnam et al., 1986; Qian & Lehman, 2018; Sleeman et al., 1986). This includes mixing up the assignment operator and the conditional equality operator (e.g., Babori, 2022) and confusing assignment in programming with mathematical ideas of equality (e.g., du Boulay, 1986). According to Qian and Lehman (2018), this last problem could lead some students to reverse the order of assignment, thinking that `5=A` is the same as `A=5`.

Some students seem to have trouble understanding variable increments such as `a=a+1` (du Boulay, 1986). Studying fifth graders in Croatia learning Python, Žanko et al. (2020) noticed that some students thought the expression above assigned the string "`a+1`" to the variable `a`, instead of incrementing the value of `a` by 1.

One of the problems with variable assignment described by du Boulay (1986) could perhaps be better explained by difficulty with the notional machine. In the example provided, some students thought that if they used the assignment `x=y` and the variable y was changed later, then the variable `x` would also be automatically changed to stay equal to `y`. This is further complicated by the fact that this "mistake" is actually valid in some cases, such as when using pointers.

6.2.6.4 Other Problems with Variables

Another issue with variables that also overlaps with not understanding the notional machine is the misconception that the name of the variable affects the variable's function (e.g., Qian & Lehman, 2018). A similar problem was seen by Žanko et al. (2020) who provides examples of students thinking that the command `print("number")` should print the value of a variable called number. However, this could also be a case of not understanding how the print statement works in conjunction with literal strings and variables, which may even be a problem with syntax.

In their list of problems for novice programmers, Garner et al. (2005) include two additional items connected to variables aside from those already mentioned. Number five on their list is problems choosing appropriate names for objects (including variables) which the authors say may indicate a lack of understanding of how the object is to be used, and number seventeen is just problems with the general concept and use of variables, a sort of catch-all for problems with variables not specifically addressed elsewhere.

Misconceptions about variables also overlaps quite a bit with misconceptions about loops. Students seem to find variables in loops generally difficult (e.g., Perkins et al., 1986), but have even more problems with the how the control variable works in `for` loops. For example, Sleeman et al. (1986) found that some novice programmers

thought that the control variable couldn't be accessed inside the loop because it didn't have a value there.

6.2.7 Loops

Loops are used in programming when a task needs to be repeated. They are one of the most important concepts in programming and, not surprisingly, cause a lot of difficulty for novice programmers (e.g., Gomes et al., 2019). Loops appear as problem number eight in Garner et al. (2005).

While the idea of repeating a task may seem simple, there are a lot of implementation details that need to be kept track of. To start with, there are a variety of different kinds of loops, each with their own syntax and conceptual requirements. To make matters worse, loop availability, implementation and syntax may differ quite a bit between programming languages.

The two most common types of loops are referred to as `for` loops and `while` loops. `For` loops generally execute a set number of times based on what is called the *control variable*. Even this aspect can look quite different depending on the language and situation. One common method is to increment or decrement the control variable after each loop iteration until a certain value is reached.

Here is a simple `for` loop in Java. Note that the control variable is declared in the `for` statement, along with its starting value, end condition and how the variable is updated at the end of each iteration.

```
for (int i=0; i<3; i++) {
    System.out.println("Monkey "+i);
}
```

Another common way `for` loops are used is to iterate through a set list of items or numbers. This is the standard method in Python as can be seen in the following code snippet which produces the same output as the Java snippet above.

```
for i in range(3):
    print(f"Monkey {i}")
```

`While` loops, on the other hand, will execute until a certain condition is met. The condition is checked at the beginning of each iteration,

including the first one. If the condition is true, the loop will execute again. If the condition is false, the loop will not execute and the program will proceed to the next line after the loop.

Here is a simple `while` loop in C++. Note that the value of `sum` will be 21 at the end of the loop since the condition is checked before the loop is executed. In other words, the last iteration will start with `total` being 15, which is less than 20, so the loop will execute one more time after this.

```
int num = 0;
int total = 0;
while (total < 20) {
    num++;
    total += total;
}
std::cout << "Sum: " << totals << " Last number added: " <<
num;
```

Here is the equivalent `while` loop in Python.

```
total = 0
num = 0
while total < 20:
    num = num + 1
    total = total + num
print(f"Sum: {total} Last number added: {num}")
```

Note that a `while` loop will check the condition before executing for the first time. It is not uncommon in programs to want the loop to always execute at least once. In Python, this problem often requires extra lines of code to solve, but some languages, such as those based on C, have a special loop called a `do...while` loop which functions exactly like a `while` loop except that the condition is checked at the end of each iteration, including the first one, instead of at the beginning. This means the loop will always iterate at least once before the condition is checked.

Some students find `for` loops easier than `while` loops (Babori, 2022), while other students think that `while` loops are easier because they have more control over them (Bosse & Gerosa, 2017). Some of the problem may be the compact syntax of these loops making them conceptually hard to understand. For example, Babori (2022) saw that

students planning their programs with pen and paper had an easier time using the conceptual idea of "repeat until" instead of `for` or `while` loops.

The creators of Scratch seem to have recognized this since Scratch provides two similar blocks which can be used for loops. The `repeat` block iterates the loop a set number of times, similar to a `for` loop, and the `repeat until` block will iterate the loop until the specified condition is met, similar to a `while` loop. However, this simplification doesn't make it easy. Studying middle school students in USA learning to program in Scratch, Grover and Basu (2017) describe students struggling to understand how loops work, and not always understanding what was included in the loop and what was outside of it. The students had even more difficulty if a variable was used for the number of iterations in a `repeat` block instead of an integer, and also had trouble understanding that a loop that contained a changing variable might do different things on each iteration depending on the value of the variable in the loop.

The literature includes a range of specific difficulties students have with loops. For example, du Boulay (1986) found that students did not understand how the control variable worked in `for` loops and did not always realize it would have a different value in each iteration. Students also thought that a `while` loop would terminate the instant the condition was met, instead of understanding that it is only checked at the beginning of each iteration. Similar problems were seen by Putnam et al. (1986) who describe students thinking that the control variable had to be changed manually inside of a `for` loop. Problems understanding the control variable are also reported by Robins et al. (2003) in their literature review.

Another common problem seen with novice programmers is understanding what is part of the loop and what is outside of the loop, especially with the statement immediately following the loops (Eckert et al., 2023; Putnam et al., 1986; Qian & Lehman, 2018; Sleeman et al., 1986).

One of the most difficult concepts with loops is nested loops (e.g., Babori, 2022; Bosse & Gerosa, 2017; Eckert et al., 2023; Perkins et al., 1986). Two loops are nested if one is inside of the other.

Here is a simple example in Python.

```
for num1 in range(3):
    for num2 in range(3):
        print(f"({num1},{num2})", end="")
```

This snippet will print out:

```
(0,0)(0,1)(0,2)(1,0)(1,1)(1,2)(2,0)(2,1)(2,2)
```

Note that the inner loop, the second number here, changes faster.

6.2.8 Conditional Statements

Both `if` statements and `while` loops use the truth value of conditional statements to control the flow of a program, another area that students struggle to fully understand (e.g., Qian & Lehman, 2018; Robins et al., 2003). Selection with `if` statements appears as problem number nine in Garner et al. (2005) and misconceptions about conditions appear as problem number ten.

Conditional statements use Boolean logic which Grover and Basu (2017) saw students struggle with when programming in Scratch. The most common problem they saw was that students treated the `or` block as if it were an exclusive or (commonly referred to as *xor* in programming). With a normal `or`, the statement is true if one or both conditions are true, but an exclusive or is only true if exactly one condition is true. In other words, it is false if both conditions are false or if both conditions are true.

In other languages, students had trouble understanding how `if` and `else` statements worked together, in particular either thinking both would be executed, or that the program would just stop if the condition failed (Qian & Lehman, 2018).

6.2.9 Structure and Planning

As should be clear by now, students struggle with a large number of specific concepts and structures used in programming. However, there is reason to believe that this might not even be the biggest trouble

students have when learning to program. For example, in a large international survey of university students learning to program, Lahtinen et al. (2005) found that understanding larger concepts connected to programming, such how to write a program to solve a task, was harder than a lot of the specific programming details. The novice programmers seemed to understand the structures in isolation, but ran into trouble when trying to put them together into a program.

This is similar to one of the main difficulties reported by du Boulay (1986), which he calls *pragmatics* and describes as the whole process of planning, creating, testing and debugging a program. Not understanding what the program is supposed to do is problem number two in Garner et al. (2005), and not being able to turn a task into a program or algorithm is problem number three.

A related problem is described by Perkins et al. (1986), who noticed that students had trouble breaking down program tasks into solvable pieces, such as identifying the need for a loop in one part of a program. Even when they were able to break the program down, they sometimes tried to do it in a way which didn't align with the capabilities of the programming language they were using. The author also reports some students have a tendency to just start writing code without taking the time to think about or plan what is actually needed.

A similar theme also appears in Robins et al. (2003), where the authors report that students spend too little time planning, create programs line-by-line instead of in conceptual blocks, spend too little time testing, and mostly make small local fixes when a problem occurs instead of rewriting larger sections of code when it might be needed.

6.2.10 What Can Be Done to Help Students Learn Programming?

There is clearly good reason to think that learning to program is not an easy task for the novice. The idea that this may be rooted in inadequate methods and unreasonable expectations has prompted some researchers to try and find ways to improve the situation instead of just identifying concepts that students struggle with.

An early attempt was made by Winslow (1996, p. 19) who lists clear steps for problem solving through programming:

1. Understand the problem,
2. Determine how to solve the problem:
    a. In some form and
    b. In computer compatible form
    (note that novices have trouble going from a to b),
3. Translate the solution into computer language program, and
4. Test and debug the program.

Many years later, Luxton-Reilly et al. (2018) reported that student self-efficacy and engagement are both important factors for succeeding at learning to program, though there is little direct advice on how to improve these factors. They also report that programming in pairs seems to lead to more positive learning outcomes than programming individually, though this effect is not seen with more experienced programmers. Positive effects are also seen from having students work out algorithms with pen and paper before starting to program. Interestingly, these algorithms do not need to be connected to the programming task.

In general, the methods discussed in the literature do not seem to differ very much from those found in other subjects such as mathematics.

6.2.11 A Note on Tinkering

Tinkering is an interesting idea that has appeared now and then in the literature and is discussed briefly below.

Perkins et al. (1986) defines tinkering as "writing some code and then making small changes in the hopes of getting it to work" (p. 48), though the authors go on to point out that effective tinkering involves reading and tracking the code to know what needs to be changed. Unsystematic tinkering can make a problem worse instead of fixing it.

A more nuanced definition was provided years later by Krieger et al. (2015) who claim that tinkering involves exploratory behaviors, deviation from instruction, lack of reliance on formal methods of learning and instruction, and the use of trial and error techniques. The authors looked specifically at gender differences in tinkering behavior

and found that women seem to tinker less and are less likely to identify themselves as a tinkerer. One explanation put forth is that tinkering is inherently risk taking behavior and that women are less inclined to take risks.

Looking at tinkering behavior in the workplace, Lyon et al. (2018) identifies two types of tinkering: testing and educated guessing, which leads to progress in learning to program, and haphazard trial and error, which does not lead to progress in learning to program.

Tinkering has also been seen with block programming (Dong et al., 2019), and a literature review of research on tinkering found an overall positive association between tinkering and learning outcomes (Timotheou & Ioannou, 2019).

From these texts, it is possible to discern that the idea of tinkering includes one of the great strengths of programing from an educational point of view: the ability to get immediate feedback on even small changes, including easily being able to test ideas with little "cost". Although this may be a natural part of working with programming tasks, especially difficult ones, a deeper dive into tinkering as a theoretical basis for these actions is beyond the scope of this text. However, it may certainly be of interest in future research to explore the connection between tinkering and completing programming tasks connected to mathematics.

6.3 Quadratic Equations

Since the interventions focus on helping students learn how to solve quadratic equations, it seems relevant to provide a thorough description of what quadratic equations are, how they are solved, and their history.

6.3.1 What Is a Quadratic Equation?

A quadratic equation is a polynomial equation in one variable, where the square of the variable appears, but no higher powers of the variable are used. A quadratic equation can always be rewritten in *standard form* as

$$ax^2 + bx + c = 0$$

where a, b, and c are *coefficients* and $a \neq 0$ (note, however, that $b = 0$ and $c = 0$ are both allowed). The left-hand side of the equation is a second-degree polynomial and the solutions to this equation are also referred to as the *roots* of the polynomial. It is common for students to start with a *monic polynomial* where $a = 1$. Note that dividing both sides by a always produces a monic polynomial without changing the solutions. This technique is used in some solution methods which only deal with monic polynomials (see below).

A quadratic equation has exactly two solutions. However, these solutions may sometimes be complex numbers, which can happen even if the coefficients are integers. For example, the solutions to the equation $x^2 - 2x + 3 = 0$ are $x_1 = 1 + i\sqrt{2}$ and $x_2 = 1 - i\sqrt{2}$. Here, subscripts are used with the variable when presenting the solutions in order to distinguish between them. This is just one of several different methods of clearly presenting the two solutions.

The two solutions may also be the same number, which makes it seem like there is only one solution, but it is more accurate to think of it as the same solution repeated twice. For example, the solutions to the equation $x^2 - 2x + 1 = 0$ are $x_1 = 1$ and $x_2 = 1$.

Although the coefficients may be any real or even complex number, it is common for students to start with integer coefficients and equations with integer solutions.

6.3.2 Solving a Quadratic Equation

A number of standard methods for solving quadratic equations are typically taught in schools, with some differences by country and by teacher. Below is a brief explanation of the most common methods used, with more detail provided for those used in this study.

6.3.2.1 Using a formula

A standard formula is perhaps the most straightforward method for solving a quadratic equation. These formulas can be given to students on a formula sheet (a common practice in Sweden today) or students can be required to memorize them (a common practice in America when I was a teenager). The formulas may be presented as is, or the teacher may go through their derivation to justify them. However,

access to a formula does not guarantee that students will be able to use it correctly (e.g., Didiş & Erbas, 2015; López et al., 2016; Olteanu & Holmqvist, 2012).

The most general formula is referred to as the *quadratic formula* in English and the *abc-formula* in Swedish. It requires the quadratic equation to first be put into standard form and then uses the following formula to find the two solutions:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

For example, using this formula to solve $3x - 12x + 9 = 0$ gives:

$$x = \frac{-(-12) \pm \sqrt{(-12)^2 - 4 \cdot 3 \cdot 9}}{2 \cdot 3}$$

This results in the two solutions $x = 1$ or $x = 3$. Note that a different method is being used here to present the two solutions. Instead of subscripts, the variable is used as is, but the two solutions are connected with the word *or* to indicate that only one solution is valid at a time. x cannot equal both 1 and 3 at the same time, it must be one or the other for the equation to be true. This may seem like a small detail, but it is an important one and is one of the common causes of student errors.

Another formula, and the one most commonly used in Sweden, is the *pq-formula*. This formula requires the left-hand side to be a monic polynomial, which can always be obtained by dividing through by a. In Sweden, an equation in this form uses different standard letters to represent the coefficients in the quadratic equation in standard form:

$$x^2 + px + q = 0$$

Equations of this form can then be solved using the pq-formula:

$$x = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}$$

For example, using this formula to solve $x - 4x + 3 = 0$ (which can be derived from the previous example equation by dividing by 3) gives:

$$x = -\frac{(-4)}{2} \pm \sqrt{\left(\frac{-4}{2}\right)^2 - 3}$$

This also results in the two solutions $x = 1$ or $x = 3$.

There is a relatively unknown version of the quadratic formula that is worth mentioning:

$$x = \frac{-2c}{b \pm \sqrt{b^2 - 4ac}}$$

Although this formula only gives one of the solutions when $c = 0$, it does give a correct answer even when $a = 0$, in which case the quadratic equation becomes a linear equation. According to Hungerbühler (2020), it is also superior to the standard formula when using approximations to numerically find the solutions.

6.3.2.2 Completing the square

This method uses standard formulas for squaring a binomial and can be used to derive both the formulas presented above. As with the pq-formula, the starting point for this method is a monic polynomial:

$$x^2 + px + q = 0$$

The next step is moving the constant to the right-hand side and then adding a new constant to both sides in order to turn the left-hand side into the square of a binomial:

$$x^2 + px = -q$$

$$x^2 + px + \left(\frac{p}{2}\right)^2 = -q + \left(\frac{p}{2}\right)^2$$

$$\left(x + \frac{p}{2}\right)^2 = -q + \left(\frac{p}{2}\right)^2$$

$$x = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}$$

For example, using this method to solve $x - 4x + 3 = 0$ gives:

$$x^2 - 4x = -3$$

$$x^2 - 4x + 4 = -3 + 4$$

$$(x - 2)^2 = 1$$

$$x = 2 \pm 1$$

And, as expected, this results in the same two solutions $x = 1$ or $x = 3$.

6.3.2.3 Factoring and the zero product property

Some quadratic equations, especially those with integer solutions, lend themselves to factoring after which solutions can be found by using the zero product property. More detail will be presented on this method as it will be the main focus of this project.

6.3.3 More on Factoring and the Zero Product Property

The zero product property appears under a variety of names. For example, it is sometimes referred to as the null product theorem (e.g., López et al., 2016), or just referred to as solving by factoring (e.g., Bossé & Nandakumar, 2005).

The zero product property is simply the fact that if the product of any two real (or even complex) numbers is zero, then one of the numbers has to be zero.

In Sweden, where this study will be conducted, using the zero product property to solve a quadratic equation is referred to as using the *nollproduktmetod*, which translates as the zero product method. Although an argument could be made that this slight change in labeling may lead to different, or perhaps additional, difficulties (e.g., more focus on the process than the properties), it seems reasonable to assume that the impact of this change of terminology will be minor and can be safely ignored for at least this study.

Given the two solutions to a monic quadratic equation, r_1 and r_2, the equation can written in factor form as:

$$(x - r_1)(x - r_2) = 0$$

Note that r_1 and r_2 can also be referred to as the roots of the second degree polynomial. The connection between solutions to polynomial equations and the roots of polynomials becomes important in more advanced mathematics.

In cases where the solutions are both integers (a common occurrence in school mathematics), this factorization can be quite easy to find given some simple techniques. Vieta's formulas for second degree polynomials can be useful here:

$$r_1 r_2 = c \qquad -(r_1 + r_2) = b$$

These formulas are easily derived by multiplying out the factors listed above:

$$(x - r_1)(x - r_2) = x^2 - (r_1 + r_2)x + r_1 r_2$$

A range of similar, though slightly more complicated, methods are available for factoring non-monic polynomials (e.g., Barry, 2008; Edmonds, 1990; Hillitt, 2007). However, this project will only deal with factoring monic polynomials.

Once the polynomial is factored, understanding the zero product property yields both solutions easily. For example, the equation $x^2 + x - 12 = 0$ can be factored to produce $(x - 3)(x + 4) = 0$, which gives the solutions $x = 3$ and $x = -4$. A common method for finding these solutions is by applying the zero product property, which states that either $(x - 3) = 0$ or $(x + 4) = 0$ and then solving for x separately in each case. This yields the two possible values of x. Another method for finding the solutions from the factored form is to understand the connection to the roots explained above, and thus that the factored equation is equivalent to $(x - 3)\big(x - (-4)\big) = 0$, which gives the solutions directly.

There are four special cases to consider even when limiting the equations to those with integer solutions. The programming tasks designed for the students in this project will not explicitly take these cases into consideration, but will allow for students to discover and explore them on their own as part of the tasks.

The first special case is not particularly interesting and can be considered a subset of the other special cases or as a degenerate case. This is the equation where both b and c are zero: $x^2 = 0$, which has two identical solutions of $x = 0$.

The second special case is where $c = 0$ but $b \neq 0$:

$$x^2 + bx = 0$$

In this case, one of the solutions is always $x = 0$. For example: $x^2 + 3x = 0$. An x can easily be factored out here to give $x(x + 3) = 0$. The zero product property can then be used to arrive at the solutions $x = 0$ or $x = -3$. In principle, this works the same as the "normal" factoring method and can be thought of as $(x - 0)(x + 3) = 0$.

The third special case is where $b = 0$ but $c \neq 0$:

$$x^2 + c = 0$$

This allows factoring by the difference of squares formula. For example: $x^2 - 9 = 0$ which can be factored to give $(x + 3)(x - 3) = 0$. Once again the zero product property gives the two solutions $x = 3$ or $x = -3$, which is sometimes written in the compact form of $x = \pm 3$.

This last type of equation can also be solved, perhaps more easily, by rewriting the equation as $x^2 = 9$ and then finding the positive and negative square root of 9, giving the solutions $x = 3$ or $x = -3$.

The fourth special case is when both solutions are identical. In this case, the polynomial can be factored into two identical factors. For example, the equation $x^2 + 6x + 9 = 0$ can be factored to give $(x + 3)(x + 3) = 0$ which is usually written as $(x + 3)^2 = 0$ and which has two identical solutions of $x = -3$.

It is worth noting that after pointing out that the majority of quadratic equations found in textbooks lend themselves to factoring, Bossé and Nandakumar (2005) show that an arbitrary quadratic equation with random coefficients is unlikely to be easy to factor. The authors use this result to argue that teachers should emphasize other solution methods in school. However, since factoring is currently an important method within school mathematics, as well as being important for more

advanced mathematics such as working with the roots of polynomials and the partial fraction method of integration, this project will focus on factoring and the zero product property as a method for solving quadratic equations.

6.3.4 A Brief History of Quadratic Equations

This section will present a relatively brief overview of the history of quadratic equations. Although some researchers have looked at the value of including lessons on the history of mathematics as part of a secondary school mathematics education (e.g., Wang et al., 2017), it may not be immediately apparent what value this information would have in this type of thesis text.

The most obvious answer is perhaps context; history is a useful tool to provide context for understanding the present. However, three of the dimensions that Wang et al. (2017) developed to study the teaching of mathematics history in Chinese high schools also seem applicable here.

- The harmony of knowledge. History helps us understand how new ideas are formed and developed and that it took time and effort to develop them.
- The beauty of ideas or methods. History can increase appreciation for beautiful new ideas.
- The charm of culture. History highlights how important mathematics is to our culture, and can help connect mathematics to other disciplines.

An interesting fact that stands out when studying the history of mathematics is how little some things have changed in the past 6000 years. Most of the early mathematical texts we have are practice problems for students, and, just like in many modern math books, most of the problems are designed to have nice numbers to make solving them easier (Friberg, 2014). As an interesting side note, and perhaps of didactical interest, Friberg (2014) explains that it is not entirely obvious how these problems were constructed with such nice numbers and this is an active field of study among some researchers. It certainly entails a deeper understanding of the mathematics to create a problem with nice solutions than to solve one.

Notable exceptions to this use of nice numbers are early Chinese texts, which, according to Katz (2009), did not shy away from including complicated problems with complicated solutions.

6.3.4.1 The Road to Irrational, Negative and Complex Numbers

A basic understanding of the history of negative and complex numbers makes it easier to understand the history of quadratic equations. For early civilizations, the only numbers that existed were positive rational numbers, in part because of the dependence of mathematics on geometry. As will be described below, this placed severe limitations on how these civilizations viewed algebra problems as well as which problems had solutions.

The ancient Greeks discovered the existence of irrational numbers over 2000 years ago (Katz, 2009). Their reliance on geometry led them naturally to these types of numbers since they appear in very basic shapes such as the diagonal of a square and the circumference of a circle. They referred to these numbers as *incommensurable*. They might say, for example, that the length of the diagonal of a square is incommensurable with the length of the side, which means there is no common unit distance that can be used to construct these lengths. However, it wouldn't be until the 9th century in the Muslim world that Abu Kamil started using irrational numbers in solutions to equations.

Some version of negative numbers, as well as mathematical rules for using them in calculations, appeared around 2000 years ago in China as a way to represent debt (Kopp, 2020). This was also seen in the 6th century in India and then several hundred years later in the Islamic world (Maryati et al., 2021). However, these numbers were still largely viewed with suspicion and were not accepted as valid solutions to equations.

It wasn't until the 16th century in Europe that Cardano started to consider negative solutions to equations, but it wouldn't be until the 17th and 18th centuries that they really started to be accepted, before which they were often referred to as false solutions or false roots (Maryati et al., 2021).

The use and acceptance of complex numbers followed shortly after the acceptance of negative numbers as solutions to equations, motivated by progress in solving cubic equations (Nahin, 2010).

6.3.4.2 The Method of False Position

Another bit of interesting background information concerns a common method used by early civilizations for solving algebra problems. This method, called *false position*, was used as far back as ancient Mesopotamia and was still seen in use as late as the 12th century by Fibonacci (Katz, 2009).

Given its success with solving linear equations, many attempts were made to apply false position to quadratic equations (Heeffer, 2014). Some success was made on some simple cases where higher powers of the variable could be treated linearly and then the answer found by calculating roots. Some more complicated cases of quadratic equations were eventually also solved by false position, which Heeffer (2014) writes was essentially done by completing the square and treating the quadratic term as being linear, and then finding the square root of the result. However, attempts to force this method to work were happening around the same time as progress was being made in algebraic notation and methods, and in the end it became much simpler to just use more standard methods for finding the roots of quadratic equations.

The basic idea of false position is to guess the answer to a problem and then check how close the guess was. The error of the guess is then turned into a scale factor which is used to scale the guess to the correct answer. False position is generally divided into *single false position* and *double false position*.

Single false position was used in ancient Mesopotamia and Egypt to solve simple linear algebra problems and was based on a single guess (Heeffer, 2014). Several example problems and their solutions are presented by Heeffer (2014). One classic example is from the Rhind Papyrus from ancient Egypt, rewritten here in more modern language: "A number plus one seventh of itself is equal to 19. What is the number?" In modern algebra this could be written as $x + \frac{x}{7} = 19$.

The solution presented by the scribe is to first guess 7 since it is easy to calculate. This gives a result of $7 + 1 = 8$. The wrong result 8 is then scaled up to the correct result 19 by the scale factor $\frac{19}{8}$ because $\frac{19}{8} \cdot 8 = 19$.

This scale factor is then used on the guess to get the correct answer: $\frac{19}{8} \cdot 7 = 16.625$, which was expressed by the Egyptians as $18 + \frac{1}{2} + \frac{1}{8}$ since they only used unit fractions.

Double false position showed up much later and was used to solve more general linear problems. It was used in 9th century Islamic mathematics and may have had connections to Chinese excess and deficit problems (Heeffer, 2014). An example of Euclid using this method to solve a problem with two unknowns is provided by Heeffer (2014):

> A mule and ass are walking together. The mule says to the ass, "If you gave me one of your sacks, I would have as many as you." The ass replies, "If you gave me one of your sacks, I would have twice as many as you." The question is, "How many sacks does each have?" (p. 67)

In modern algebra this could be written as $m + 1 = a - 1$ and $a + 1 = 2(m - 1)$. Using double false position a good first guess, or first position, might be that the mule has 4 sacks, which would mean that the ass has 6 sacks by the first rule. But by the second rule, the ass would have $2(4 - 1) - 1 = 5$ sacks. This gives an error of $6 - 5 = 1$ (note that the error here is a straight difference and not a proportion). A second guess, or second position, might be that the mule has 8 sacks, which would mean that the ass has 10 sacks. But by the second rule the ass should have $2(8 - 1) - 1 = 13$ sacks. This gives a new error of $10 - 13 = -3$.

The double false position calculation is more complicated and involves multiplying each position's error by the other position, and then dividing the difference of these values by the difference of errors. In modern notation this would be: $\frac{(1 \cdot 8) - (-3 \cdot 4)}{1 - (-3)} = 5$. Which means the correct answer is that the mule has 5 sacks and the ass has 7 sacks. Note that the example guesses were provided by Heeffer (2014) and were not directly attributed to Euclid, which may explain the presence of negative numbers in the calculation.

6.3.4.3 Freeing Algebra From Geometry

Although it is not so easy to define what, exactly, algebra is (Katz & Parshall, 2014), most people would probably agree that quadratic equations are part of algebra. As such, the history of quadratic equations is inexorably tied up with the history of algebra as a whole.

There are a number of different ways of categorizing the history of algebra. One method seen in the literature is to divide it into three stages (Katz & Barton, 2007):

- The rhetorical stage where all the arguments are explained with common language.
- The syncopated stage where some symbolic abbreviations start to be used.
- The symbolic stage where algebra is expressed by set of commonly used symbols.

However, an alternate categorization is suggested by Katz and Barton (2007) where they divide the history of algebra into four conceptual stages:

- The geometric stage where almost all ideas and arguments are based on geometry.
- The static equation solving stage where the focus was on finding numbers that satisfy different relationships.
- The dynamic function stage where the idea of motion and physics started to play an import part in the study of algebra.
- The abstract stage where the focus was on studying algebraic structures themselves.

On a broader level, the history of algebra could perhaps also be described as a long process of freeing itself from geometry. Almost all early algebra was based on geometry, sometimes referred to as geometric algebra (Allaire & Bradley, 2001). In geometric algebra, not only are solutions based on geometry, but even numbers have to be represented by physical objects such as the length of a segment or a rod, or the area of a square. This is why, according to Allaire and Bradley (2001), second and third powers are referred to as squares and cubes in English.

Tying quantities to physical objects like this limited them to positive numbers, a limitation that took thousands of years to overcome. Tying algebra to geometry more or less limited mathematicians to finding solutions to cubic equations, with the obvious exception of some basic cases involving higher degree polynomials and monomials (Baga, 2017).

The oldest mathematical texts we have today are from the proto-literate period in Mesopotamia and date back to around 3000 BCE (Friberg, 2014). An example problem from one of these earliest tablets is asking a student to find the area of two quadrilateral fields given their sides.

The first texts that give rise to quadratic equations are Old Akkadian texts from around 2300 BCE. A common geometric method used to solve these problems using successive approximations, as explained by Friberg (2014), was the field expansion procedure which involved successively adding unit fractions until a goal was reached. For example, one text contains a problem asking the reader to find the length of a line in a trapezoidal field that is parallel to the base and divides the area in half .

Babylonian tablets from around 2000 BCE are a rich source of mathematical history and include solutions to simple linear equations as well as systems of equations (Katz, 2009). These tablets include a variety of numerical algorithms, based on geometry, including some for finding square roots and reciprocals of integers (Allaire & Bradley, 2001). According to Friberg (2014), reciprocals were important enough for the Babylonians to create tables of them to be used in other problems. For example, Katz (2009) writes that they solved equations of the type $ax = b$ by multiplying by the reciprocal of a.

The Babylonian texts also contain problems that give rise to quadratic equations. Because they only worked with positive rational numbers, they divided quadratic equations into nine different types (Baga, 2017). Note that even if the Babylonian solutions were based on geometry, they often focused on presenting numerical algorithms for them.

One type of equation was $n + \frac{1}{n} = k$, for a given k. Using modern methods, this equation could be multiplied by n and rearranged to get the following quadratic equation: $n^2 - kn + 1 = 0$.

According to Friberg (2014), the Babylonians were able to use the following identity, together with their tables of reciprocals, to solve these types of equations:

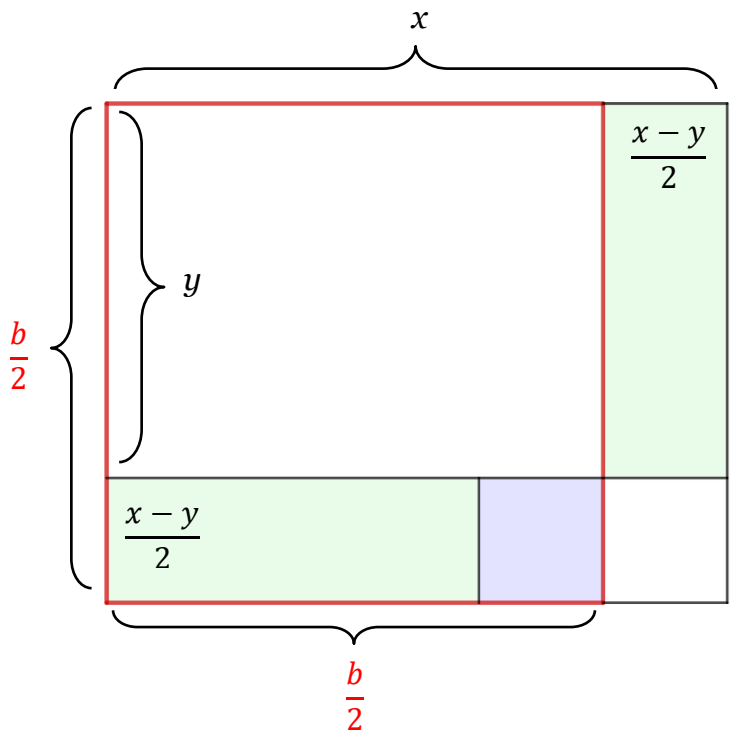
$$\left(\frac{1}{2}\left(n + \frac{1}{n}\right)\right)^2 - 1 = \left(\frac{1}{2}\left(n - \frac{1}{n}\right)\right)^2$$

This identity allows you to treat $n+\frac{1}{n}$ as one number and then find $n-\frac{1}{n}$. Adding these two numbers together and dividing by two gives you n.

Another type of quadratic problem found on the tablets involved finding the sides of a rectangle given the semiperimeter (half the perimeter) and area (Katz, 2009). In modern notation, this would be solving the system of equations $x+y=b$ and $xy=c$. A modern explanation for their solution involves assuming $x>y$ and noticing that $\frac{b}{2}=\frac{x+y}{2}=x-\frac{x-y}{2}=y+\frac{x-y}{2}$.

Figure 1

Babylonian Method for Solving $x+y=b$ and $xy=c$



Start by constructing a square from half the semiperimeter, $\frac{b}{2}$; this gives the red square in Figure 1, with area $\left(\frac{b}{2}\right)^2$. Subtract the given area, $xy=c$, from this. Notice that the given area is the large white rectangle plus a green rectangle in figure 1. If we move this green rectangle to the bottom we can easily see that what is left over is the little blue square

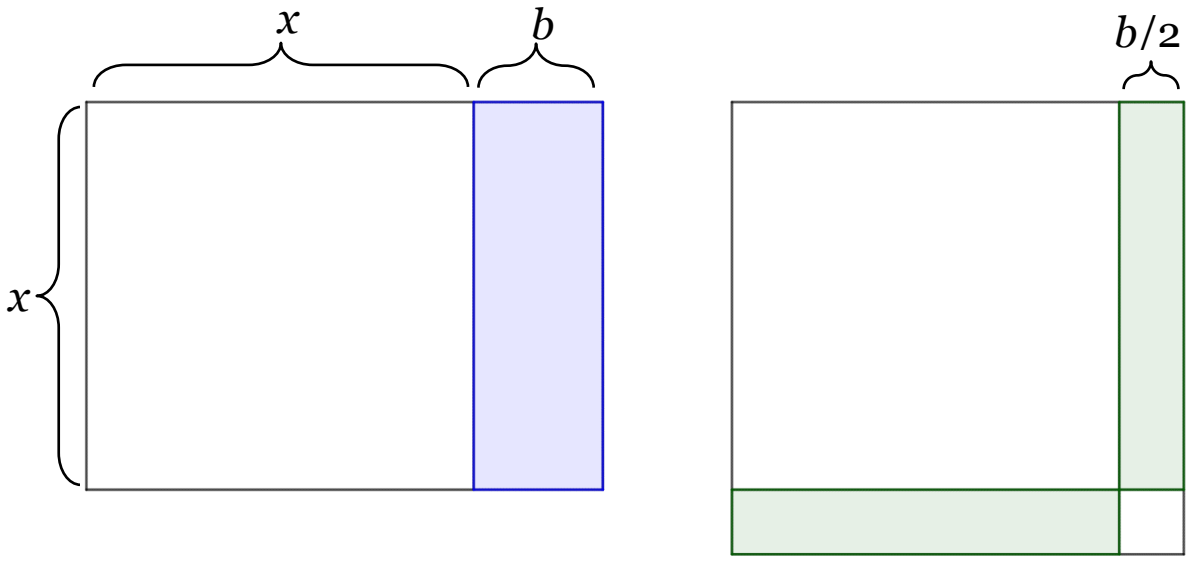
with area $\left(\frac{x-y}{2}\right)^2$. Take the square root of this to get $\frac{x-y}{2}$, which, when added to $\frac{b}{2}$, gives x.

Other quadratic equations they presented geometric solutions for include $x^2 \pm bx = c$, which were considered two different types since b was always a positive number (Katz, 2009). The following is a modern interpretation of the methods given by the Babylonians for solving $x^2 + bx = c$ which is essentially an early version of the method of completing the square.

Start by drawing a square with sides x and then extend from it a rectangle of width b as seen in the first image in Figure 2. Then move half of the blue rectangle to the bottom of the square as seen in the second image. Note that this now creates a new square with sides $x + \frac{b}{2}$. Also note that the area of this new square is slightly bigger than the original area since we have added a new small white square with area $\left(\frac{b}{2}\right)^2$ to the original area which was c.

Figure 2

Babylonian Method for Solving $x^2 + bx = c$



This gives us the new equation $\left(x + \frac{b}{2}\right)^2 = c + \left(\frac{b}{2}\right)^2$. This equation can now be solved by taking the square root to get $x = \sqrt{\left(\frac{b}{2}\right)^2 + c} - \frac{b}{2}$.

Although quite a bit of mathematics has been found on ancient Egyptian papyrus scrolls from around 1800 BCE, Katz (2009) points out that they only had one example of a quadratic equation arising from dividing a square area into two smaller squares with a given ratio of areas.

The ancient Greeks, starting with Euclid around 300 BCE, had an even stronger geometric basis for their algebra. Numbers were represented by the length of line segments and the area of figures such as squares (Katz, 2009). Euclid presented purely geometric solutions to geometric problems which were essentially quadratic equations. An example provided by Katz (2009) is Proposition VI-29 in Euclid's Elements which presents a construction for $x = \frac{b}{2} - \sqrt{\left(\frac{b}{2}\right)^2 - c}$ as the solution to $bx - x^2 = c$.

In the 3rd century CE, Diophantus made some progress in freeing algebra from geometry by using symbols to describe different types of linear and quadratic equations and their solutions. An example problem was finding an integer n such that $18 + 3n - n^2$ is a positive square number (Katz, 2009).

Progress in algebra was also made in China with the book *Nine Chapters on the Mathematical Art*, which was written over several centuries from around 200 BCE to the 2nd century CE. This book contained algorithms for solving systems of linear equations that don't seem to be based on geometry. However, Katz (2009) points out that there were only hints of solutions to quadratic equations in China during this time.

Jumping to the 5th century in India, Katz (2009) describes evidence appearing there of rules for solving quadratic equations. One example of the use of quadratic equations at this time was to solve problems involving arithmetic progressions.

In India during the 6th century, Brahmagupta presented the following three kinds of quadratic equations and their solutions: $a^2 + bx = c$, $bx + c = ax^2$, and $ax^2 + c = bx$. Although he made progress on quadratic equations, Brahmagupta only recognized single solutions to them (Katz, 2009).

Brahmagupta did show some recognition of negative number, but it wouldn't be until the 7th century that Indian mathematicians started to really recognize the existence of negative and irrational numbers as well as that quadratic equations have two solutions. This, together with their decimal system and use of 0, allowed them to make great progress in developing algebra (Baga, 2017).

A significant step in freeing algebra from geometry, and developing what some researchers refer to as arithmetical algebra, was made in the Islamic world by al-Khwarizmi in the 9th century (Baga, 2017). al-Khwarizmi started with arithmetical solutions to quadratic equations and only gave geometric justifications afterwards.

Negative numbers were still not accepted, so al-Khwarizmi divided quadratic equations into the following five types: $x^2 = bx$, $x^2 = c$, $x^2 = bx + c$, $x^2 + c = bx$, and $x^2 + bx = c$ (Allaire & Bradley, 2001). He used what we would today call completing the square to solve the last three types.

It is worth noting that his texts contained mostly theoretical number problems and very few practical examples, unlike most of the earlier traditions (Katz, 2009).

The next few hundred years saw more progress on algebra in the Islamic world, including, according to Katz (2009), a better acceptance of square roots as numbers, the use of negative coefficients with polynomials, and the development of the laws of exponents. Progress was also made in China at this time and there is textual evidence for a flow of mathematical ideas between China, India and the Islamic world around the 11th and 12th centuries (Chemla, 1993).

Because of the introduction of Hindu-Arabic numbers and cheap paper to Europe in the 14th century, Katz (2009) argues, textbooks containing algebra problems that could be solved with pen and paper by students became more common. The next few hundred years in Europe saw improvements in the use of algebraic symbols, moving it even further away from geometry, as well as further acceptance of negative numbers and the beginning of the idea of complex roots (Katz, 2009).

In 16th century Europe, Cardano started to consider negative numbers as solutions to equations, and even, to some degree, their square roots, but he had trouble accepting them, calling them *false solutions*. Full acceptance of negative and imaginary solutions was still a long way off (Corry, 2015).

The fundamental theorem of algebra first appeared in the 17th century, as well as the first version of the modern factor theorem for polynomials (Katz, 2009). In simple terms, the fundamental theorem of algebra states that a degree n polynomial has exactly n roots (counting multiple roots as distinct). Also in simple terms, the factor theorem states that if r is a root of a polynomial $p(x)$, then $(x - r)$ is a factor of $p(x)$.

It is worth noting that even as algebra was moving further and further away from geometry at this point, it was during the 17th century that Descartes and Fermat made a strong connection between algebra and geometry through the development of coordinate systems. However, their efforts should not be seen as a step backwards, as they did not increase algebra's dependence on geometry but rather allowed solutions from one to be applied to the other. For example, Descartes showed how it was possible to use algebra to solve geometry problems (Katz, 2009).

Numerical methods for calculating the roots of polynomials were developed in the 18th century, and, according to Katz (2009), the first rules for working with imaginary numbers.

In the 19th century, algebra started to change from just being about solving problems to the study of structures based on well-defined axioms, including theories about groups, rings and fields. It was here, Katz (2009) writes, that Hamilton was able to create a description of algebra that included negative and imaginary numbers.

This brings us to modern times, with our current symbolic representation for algebra and the full acceptance of and rules for negative numbers, irrational numbers and complex numbers, as well as the continued study of algebraic structures.

6.3.4.4 Modern Advances in Quadratic Equations

Although many people may assume that there is no room left for progress on solving quadratic equations, two new insights have appeared in recent years.

Starting from a normalized monic quadratic equation $x^2 + bx + c = 0$, Loh (2019) rewrote the roots based on their average as $r_{1,2} = -\frac{b}{2} \pm u$. From this he gets $u = \sqrt{\frac{b^2}{4} - c}$ by multiplying the roots, which he claims may be more intuitive since there is no need to complete the square or memorize a formula.

Another new formula was presented by Hungerbühler (2020) who started with what he calls the homogenous and normal form of a quadratic equation $x^2 - 4ux + v^2 = 0$, with symmetric solutions $r_{1,2} = \left(\sqrt{u+v} \pm \sqrt{u-v}\right)^2$. Notice that in this case b has been replaced by -4u and c has been replaced by v^2. Although this may require finding a square root to determine v, Hungerbühler (2020) argues that many problems from physics and geometry naturally lend themselves to this form.

6.3.4.5 Returning to Geometry

Although the lack of dependence on geometry allowed great progress to be made in developing and understanding algebra, there is some evidence that algebra may have moved too far away from geometry, especially in the minds of students. For example, Lemańska et al. (2014) presented first year university students with problems which were simple to solve with geometry but much more difficult using only algebra. They found that very few of the students even considered using geometry as a solution method. As a result of their findings, the authors suggest teaching a stronger connection between algebra and geometry in schools.

7 Theoretical Background

This section explores the important theoretical concepts used in this thesis.

7.1 Proceptual Thinking

As can be seen from the literature, a common problem students have when working with quadratic equations is an over reliance on procedures. Here it will be useful to introduce the idea of *proceptual thinking*, put forward by Gray and Tall (1994). Before introducing their idea, the authors first engage in a thorough discussion of the differences between procedural thinking and the more flexible and desirable conceptual thinking, while at the same time acknowledging evidence that good procedural fluency may contribute to the development of conceptual thinking.

As noted by Gray and Tall (1994), mathematicians and other people with good mathematical skills often find it easy to use mathematical notation in an ambiguous manner, changing interpretation fluidly from moment to moment to suit their needs. This is of particular interest when the same notation represents both a procedure and a concept at the same time. For example, $\frac{13}{7}$ represent both the procedure of dividing 13 by 7 as well as the solution to that calculation when expressed as a rational number. The authors refer to this type of flexible thinking as proceptual thinking, where the term *proceptual* is a portmanteau of the words “process” and “concept”. They believe that this type of flexible thinking is at the root of fluency in mathematics, and the lack of it is a source of constant problems for students who struggle with mathematics. They refer to this difference as the *proceptual divide*.

Further support for the existence of a proceptual divide was found by Prayogo et al. (2020) when looking at lower secondary school students' ability to solve mathematical problems. The authors found that the students who had an easier time solving the problems also had an easier time with proceptual thinking.

Using the idea of proceptual thinking in algebra, Tall (1992) reports on an experiment using computer programming to try and help students

see algebraic expressions as both an object (concept) and a process (procedure). The author's idea was that letting the computer do the calculations would help focus the students' attention on the expressions as objects and leave them more free to consider and reason about properties of the expressions. The students who worked with the computer program were found to develop a more flexible and meaningful understanding of algebraic expressions.

Theories connected to procepts also played an important role when Kidron (2008) looked at ways to help first year university students better understand limits. In her study, Kidron (2008) combined the use of procepts with the theory and methodology Abstraction in Context, to analyze the results of using computer aided tasks in helping students understand the concept of limit (see section 8 for more information about Abstraction in Context). The author stated that viewing student misunderstandings through the lens of proceptual thinking was key to gaining insight into why they were having difficulties and writes "The procept theory is used as a cognitive frame but we may also discern its strong epistemological basis: processes and objects are mathematical constructs as well as cognitive entities" (p. 199).

The lack of proceptual thinking can be seen in the literature as one interpretation of the errors students make when trying to solve quadratic equations, stemming from too much focus on memorizing procedures that that they have difficulty mastering. It follows that trying help students develop their proceptual thinking may help them avoid many of these commonly reported errors. As already shown by Tall (1992), computer programming may be a good way to accomplish this goal.

7.2 Creative Mathematical Reasoning

Another take on the problem with students focusing too much on procedures and not enough on understanding is presented by Lithner (2008) when he discusses the differences between what he refers to as imitative mathematical reasoning and creative mathematical reasoning. Lithner makes it clear that imitative reasoning is less desirable and does not lead to or reflect a deeper understanding of mathematics, while creative mathematical reasoning does.

Lithner (2008) presented the following criteria for creative mathematical reasoning (p. 266):

1. Novelty. A new (to the reasoner) reasoning sequence is created, or a forgotten one is re-created.
2. Plausibility. There are arguments supporting the strategy choice and/or strategy implementation motivating why the conclusions are true or plausible.
3. Mathematical foundation. The arguments are anchored in intrinsic mathematical properties of the components involved in the reasoning.

The author also divides up imitative reasoning into subcategories which he then explains in depth. He refers to one of these categories as *algorithmic reasoning*. In this type of reasoning, students rely on algorithms (either from guides or memorized) as a method of avoiding actual mathematical reasoning. What makes this interesting in the context of this thesis is that algorithmic thinking is often associated with computational thinking, programming and the use of computers to solve problems in general (e.g., Dimos et al., 2022; Kallia et al., 2021), and is even explicitly included as a central content in the Swedish mathematics curriculum for grades 4-9.

On the face of it, this may seem to imply that programming may not be a great way to help students understand mathematics. However, a careful reading of the different views on algorithmic thinking reveals subtle differences in what is being discussed by Lithner compared to the authors discussing algorithmic thinking, especially in the context of programming and computational thinking. Lithner (2008) is clear that the type of imitative reasoning that he calls algorithmic reasoning involves recalling or referencing a previously known algorithm, also sometimes referred to as a procedure. This is not really the same as the more positive view on algorithmic thinking as a way of breaking down a complex problem into a series of smaller steps, where a known method or algorithm can be applied to each smaller step, similar to the idea of problem decomposition described by Pólya (1945) and which is still used in mathematics and computer science education research today (e.g., Prabawa et al., 2023). In this later view, it seems implicit that the overall algorithm is previously unknown to the students, even

if the algorithm for each small step might be. If this is truly the case, then this type of algorithmic thinking may be better classified as a mix of creative mathematical reasoning and imitative reasoning.

Lithner (2008) articulates a number of ways students tend to shortcut tasks that intend to promote creative mathematical reasoning and end up engaging in imitative reasoning instead, including trying to copy the procedure shown in a book or a teacher's example, or misapplying an inappropriate algorithm that seems right. He makes the additional point that this is often overlooked as many teachers often accept even shallow attempts at reasoning as valid. The risks with not being aware of this problem were highlighted by Schoenfeld (1988) when he carefully examined a mathematics lesson that many would consider to have been well-taught. After a careful analysis, Schoenfeld (1988) saw evidence that the students were still mostly learning a procedural type of thinking and not learning a deeper understanding of the relevant mathematics.

Well-designed programming tasks can hopefully help prevent a lot of these problems; the program simply won't work if the wrong procedure is applied, and the act of programming a general solution often inherently requires a deeper understanding of a procedure instead of just copying it.

7.3 Competing Views on Learning

An issue which can sometimes cause problems in mathematics education research is the often acknowledged polarization between cognitive and sociocultural theories. A good summary of these two views is provided by Danish and Gresalfi (2018).

In short, cognitive theories focus on the idea that learning happens in the individual’s mind, and it is there that we should look for signs of it. Sociocultural theories, on the other hand, focus on the idea that learning is a social construct which involves the participation in cultural practices and cannot be meaningfully separated from its context. The differences between these viewpoints often give rise to various critiques from those arguing for their own stance:

> Cognitive perspectives are critiqued for their focus on individual characteristics within experimental contexts, thus missing or ignoring details

> that reflect the real-world links between individuals and their context. Sociocultural perspectives are critiqued for their focus on context, making it difficult to produce any systematic, actionable, and generalizable results; they often lose the individual due to the focus on the collective. For many, these differences are irreconcilable. (Danish & Gresalfi, 2018, p. 37)

These differences can also be looked at as competing metaphors for learning: learning as acquisition (cognitive theories), where humans gain ownership over something called *knowledge*, compared to learning as participation (sociocultural theories), where learning is the process of becoming a member of a specific community (Sfard, 1998).

However, not everyone accepts the necessity of the aforementioned dichotomy in theoretical positions. A number of researchers made early attempts at arguing for the necessity and value of both outlooks and even discussed ways to reconcile them (e.g., Cobb, 1994; Sfard, 1998).

Building on the works of Hans Freudenthal, Adrian Treffers and Vasily Davydov, Abstraction in Context is another attempt at designing a theoretical framework that combines both cognitive and sociocultural ideas (Dreyfus et al., 2015; Dreyfus & Kidron, 2014). Abstraction in this theory is based on the theory of *vertical mathematization* (in AiC changed to *vertical reorganization*) developed by Hans Freudenthal (1973) and Adrian Treffers (1987), as well as Vasily Davydov's *ascent to the concrete*. AiC then examines these ideas of abstraction situated in a particular social context (Dreyfus & Kidron, 2014).

7.4 What Is Abstraction?

Abstraction is a word that appears frequently in mathematics education and mathematics education research. When examined closely, the idea of abstraction is not so easy to pin down and a common colloquial understanding of the word is not always useful in mathematics education research.

Broadly speaking, there are two main views of abstraction commonly used in mathematics education research, each of which is connected to a different set of ideas on how students should learn mathematics.

The perhaps more prevalent idea is based on Jean Piaget's view that normal development proceeds from the concrete to the abstract (Coles & Sinclair, 2019). Here, concrete is often, though not necessarily,

connected to perceiving something with the senses, such as using manipulatives. Abstraction is then considered a subtractive process of decontextualization away from the concrete and towards more general ideas such as symbol systems or equivalence classes (Coles & Sinclair, 2019; Harel & Tall, 1991; Hershkowitz et al., 2001). Note that in these views, the idea of *abstractness* seems to be an inherent property of the objet or construct itself and, according to Coles and Sinclair (2019), is based on an acquisitionist view of learning.

Abstraction can also be thought of as relational, an idea connected to the works of Gottlob Frege, Lev Vygotsky and Vasily Davydov (Coles & Sinclair, 2019; Hershkowitz et al., 2001; Scheiner, 2016). Here, abstract is not an inherent property of an object or construct, but instead a person's relationship and understanding of the object or construct. Vasily Davydov's theory of *ascent to the concrete* reverses the order of learning: students start with an undeveloped form and then move towards a more developed and consistent form (Hershkowitz et al., 2001). Here learning is a recontextualization as opposed to the previously described decontextualization.

Another theory of abstraction that seems to have connections to both the aforementioned points of view was adapted from computer science education research by Rich and Yadav (2020). In this theory, abstraction is connected to how detailed an idea is, where less detailed means more abstract. Interestingly, the authors make the case that there is no correct order of learning, but rather the ability to move fluidly up and down the different levels of abstraction is the key to being able to effectively solve problems within mathematics.

This thesis will use the definition of abstraction in Hershkowitz et al. (2001): "an activity of vertically reorganizing previously constructed mathematics into a new mathematical structure" (p. 202). Here, *vertically reorganizing* is a renaming of Treffer's (1987) *vertical mathematizing*, where mathematizing is understood to be the process of students re-discovering or re-inventing mathematics on their own through working with various tasks and activities (Schwarz et al., 2009). Vertical mathematizing is the process of building new mathematical knowledge upon the structures of prior mathematical knowledge. This is in contrast to horizontal mathematizing, which is

the process of using mathematics to model and describe some aspect of reality (Treffers, 1987).

Using this definition, Hershkowitz et al. (2001) explain that the process of abstraction is an activity that takes place in a context and which requires the use of theoretical thinking, although it may include empirical thinking as well. This process is socially situated and is influenced by tasks, tools and artifacts, and depends on the personal history of the students as well as the teacher.

To add clarity to their discussion, the authors define and discuss the terms *activity* and *context* as well. An activity is a connected sequence of events with shared content. Activities always include artifacts which mediate actions in the activity. Context is the sum total of all the factors that frame a person's actions. Context can be external, such as the room someone is working in or the tools they are using, but also includes personal history, social relationships, existing mental constructs and emotional states. In this sense, an activity and its context cannot be separated.

7.5 Algebraic Thinking

Algebraic thinking is a broad topic in modern research, and it is beyond the scope of this text to provide a comprehensive discussion of the topic. However, one specific theoretical framework within this field may prove to be useful when analyzing this study's data, namely the three layers of algebraic thinking introduced by Radford (2006): *factual, contextual* and *symbolic*.

Although the author developed his idea in response to studying how students generalized patterns, the layers themselves are useful descriptors for other areas of mathematics connected to algebra. Radford (2006, p. 15) suggests that moving through these layers is "about gaining access to deeper forms of consciousness." As such, it may work well with the other theoretical frameworks used in this text to help shed light on the construction of new knowledge.

In the *factual* layer, students are limited to discussing the specifics in front of them. They do not discuss or even name any indeterminate ideas or objects.

In the *contextual* layer, students start to name the indeterminate, but it is still tightly bound to the context. The author provides the example of students referring to "the next figure" or "the top row".

In the *symbolic* layer, students are able to work with algebraic symbolic representations independent of the context. It is important to note that the expressions and equations used are still derived from the context and any solutions or results attained must again be reintegrated into the context to make sense.

This final symbolic layer of being able to smoothly shift between contextual and symbolic ideas can also be seen as a type of proceptual thinking, where any equations are both conceptual objects representing a context as well as objects that can be manipulated according to a known set of rules and procedures.

8 Abstraction in Context: A Theory and a Methodology

This research project will rely heavily on the theoretical framework Abstraction in Context (AiC). This framework provides a method for identifying students' epistemic actions as evidence of knowledge construction.

AiC has been used in a number of studies looking at how students learn mathematics. For example, it was used to study a Czech student's construction of internal mathematical structures by Stehlikova (2003). In Turkey, Ozmantar (2004) used AiC to study how two 17-year-old students understood the graphs of the absolute values of linear functions. Kouropatov and Dreyfus (2017) combined AiC and proceptual thinking (Gray & Tall, 1994) to study how upper secondary school students in Israel learned about integrals. AiC was used in Indonesia to analyze how preservice mathematics teachers understood parallel coordinates (Nurhasanah et al., 2017) and to analyze the abstraction process of 8th graders working with tangents to circles (Nurhasanah & Sumekar, 2019). In Israel, Tabach et al. (2017) combined AiC with another theory called Documenting Collective Activity to study 8th grade students working with probability.

AiC has even been used to look at teaching mathematics through programming. For example, Anabousy et al. (2023) used AiC to study how programming with Scratch might help Israeli 6th graders learn about coordinate systems.

Abstraction plays a central role in AiC. As pointed out by Hershkowitz et al. (2001), abstraction is a mental activity, and as such, it cannot be observed directly and instead requires theoretical help to observe. The authors developed AiC to solve this problem by looking for epistemic actions, which are "mental actions by means of which knowledge is used or constructed" (p. 203). Unlike the process of abstraction, epistemic actions are observable by the participants and the researcher and thus it is possible to look for relevant epistemic actions as indications of the formation of new abstractions.

AiC specifies a set of epistemic actions that the researcher can try and identify, making it both a theoretical framework as well as a methodology.

Note that in this text "knowledge construct" and "knowledge element" will be used synonymously, mainly for reasons connected to the flow and readability of the text.

8.1 The Frist Step

The first step in the AiC methodology is an a priori analysis of the proposed activities based on students' assumed previous knowledge as well as which new constructs they may need to develop in order to complete the activities. The following is a summary of this process which is explained in detail in Dreyfus et al. (2015).

- Make assumptions about the students' relevant prior knowledge, preferably in conjunction with a teacher who is familiar with the students' backgrounds.
- Consider the knowledge elements the researcher and teacher intend the students to construct during the activities.
- Consider what the students need to know in order to complete the designed activities in such a way that they will construct the desired knowledge elements.
- Intended knowledge elements should have operational definitions and not just mathematical ones so that they can be clearly identified by the researcher.
- The analysis will not just produce a list of knowledge elements but will include dependencies as well since most knowledge elements will depend on other knowledge elements.
- An overarching knowledge element should be identified as the main goal of the activities.

The authors point out that during the analysis of the results, it is important to distinguish between new constructs intended by the researcher and unintended new constructs which may be identified during the a posteriori analysis (see below).

8.2 RBC+C Model

After the a priori analysis, the students work in groups on the tasks and are recorded in some fashion as they work. Afterward, when all the tasks have been completed, the recordings are analyzed using the RBC+C model (Dreyfus et al., 2015). This model is used to try and identify the emergence of new mathematical constructs using the epistemic actions of recognizing, building-with, constructing, and

consolidating. These epistemic actions are nested, meaning that some actions are dependent on others, as described below:

- **R**ecognizing: the student recognizes previous constructs that may be of use in the task. This can usually be identified by a specific action a student undertakes or when they mention a previous construct. The authors claim that identifying recognizing actions is quite easy and go on to emphasize that this does not refer to any new constructs but only to older ones already available to the student.
- **B**uilding-with: the student uses the recognized previous constructs in the task. For example, the student may use the previous construct to perform a calculation, draw a picture, motivate a choice or as the basis for a way of reasoning. Here again, the authors claim that building-with actions are easy to identify. Building-with actions depend on recognizing actions since a construct needs to be recognized before it can be used.
- **C**onstruction: the student starts to develop a new construct. In this framework, constructing is considered the central epistemic action for mathematical abstraction. Unlike the recognizing and building-with actions, identifying the constructing actions can be quite complicated. The authors point out that the indications of constructing can be quite spread out through the activity. Constructing actions depend on recognizing and building-with actions.

The final C in the methodology stands for *consolidating*, which is a long-term and deeper form of construction and usually happens when the student can use the new construct as a building block for future constructs. When a new construct first emerges in a student's mind, they may not be consciously aware of it yet, or they may not have a very firm grasp of it. It is not until it can be recognized and used as part of a build-with action for a new construct that it is considered to be consolidated.

There does not seem to be a good theoretical basis for identifying when knowledge is actually consolidated. Because of this, no attempt will be made to identify consolidating actions in this thesis, and so all coding using **C** will only refer to construction actions.

8.3 Shared Knowledge

Assessing the construction of new knowledge in groups is inherently more difficult than in an individual, though it is important to understand the connection between the two in research involving students working together in groups (Hershkowitz et al., 2007). The problem is exacerbated by practical difficulties in collecting and analyzing group data, such as the time required to sift through large amounts of data or the difficulties in understanding the contributions of group members who do not communicate in obvious ways.

In order to adequately apply the RBC+C model to group work, Hershkowitz et al. (2007) have extended the theory to include a group's construction of *shared knowledge*, which the authors define as "a common basis of knowledge, which allows the students in the group to continue together the constructing of further knowledge in the same topic" (p. 4).

The authors discuss the importance of focusing on the flow of knowledge within the group and highlight the following points for use when identifying and analyzing shared knowledge within the AiC framework:

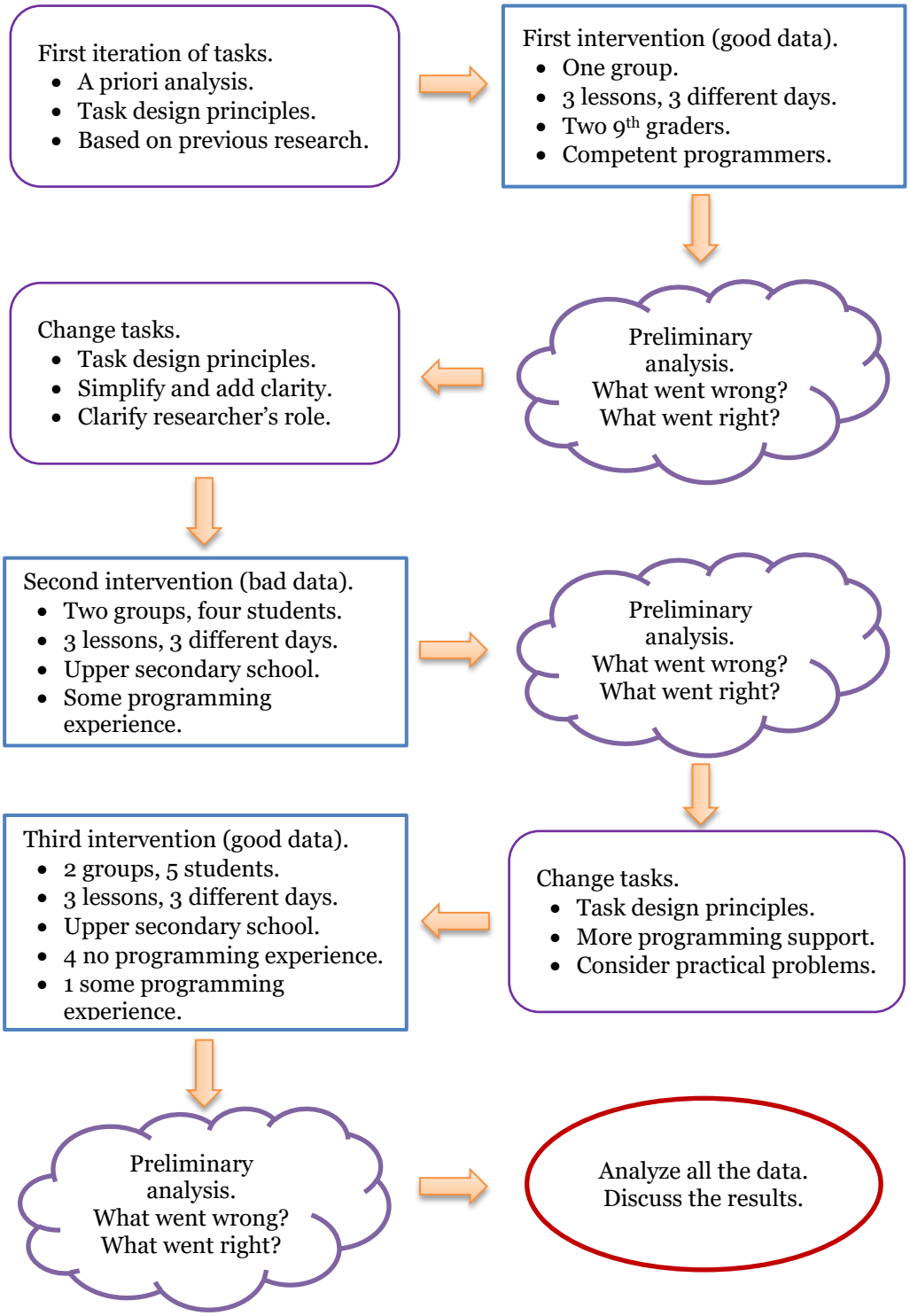
- The researcher should look at the individual's contribution to the group's construction of shared knowledge.
- The researcher should look for the group constructing knowledge in one lesson and then using it another lesson to identify cases when the group consolidates the knowledge they constructed.
- The researcher should use the theories of abstraction that underpin AiC to identify vertical mathematization.
- The researcher should use the RBC+C model to analyze when a group constructs shared knowledge.

9 Method

Data for this study was collected in three separate task design and intervention cycles. Figure 3 provides an overview of this process.

Figure 3

Overview of the Intervention Process



Each intervention was guided by the task-based interview methodology (Maher & Sigley, 2020). Data collection proceeded similarly in all three interventions, although there were some important differences (discussed in detail below). In all cases, an Olympus digital voice recording device was used to capture audio, and some form of screen recording software was used to record programming work done during the lessons. All written work produced by the students was also collected after each lesson.

Each intervention consisted of three lessons on three different days where the students worked in groups. Each group would try and complete a series of tasks, some with pen and paper and some by writing a program.

After a group completed a task, I, acting as the teacher, would check the results, ask a few clarifying questions, and, if satisfied with the result, would give them the next task. The students were told they could ask questions and that I would provide as much help as I could.

The AiC methodology was then used to analyze the data from each intervention. Three additional theoretical frameworks were used during the analysis in order to more effectively highlight important aspects of student reasoning: proceptual thinking (see section 7.1), creative mathematical reasoning (see section 7.2), and three levels of algebraic thinking (see section 7.5).

Each intervention is described in detail below in its own section, including a more detailed description of the participants.

9.1 Task-based interviews

Task-based interviews are defined by Maher and Sigley (2020) as interviews "in which a subject or group of subjects talk while working on a mathematical task or set of tasks" (p. 821). In this study, this was facilitated by recording the students while they worked in groups to complete tasks. Acting as both teacher and researcher, I would ask pertinent question to help them vocalize their thought processes, while also answering questions they had about the tasks. As Maher and Sigley (2020) explain, this methodology allows the researcher to employ a variety of techniques, such as "thinking aloud" and "open ended

questions" to help elicit responses from the subjects, both of which were encouraged by the group work and the task design.

9.2 A Priori Analysis

Ideally, when using the AiC methodology, an a priori analysis of the students' assumed prior knowledge is done in conjunction with the participating teachers in order to tap into the teachers' knowledge of their own students. However, participating teachers lacked the time to participate in the interventions to the extent originally hoped for and recommended in the AiC framework. In addition to lack of time, some of them also felt they had insufficient programming experience to lead the lessons. Because of this, it was decided that I would act as the teacher in all interventions, including handling the a priori analysis alone. The basis of the analysis in this case was the current literature on quadratic equations, the typical level of knowledge expected for students in Sweden in different years, as well as a brief discussion with the participating teachers about the particular students. In addition to the information about the most recent grade the students received in mathematics, the teachers were also asked for details on how much the students knew about quadratic equations, how good their basic algebra skills were, whether or not they could multiply two binomials, and, of course, their previous programming experience.

9.3 Task Design

Task design is a fundamental aspect of this project. In their work designing a mathematics curriculum for computerized environments, Hershkowitz et al. (2002) identified several important components of mathematical tasks. The following are the ones most relevant for this project:

- The students should be asked to discuss and predict how the tasks may be solved before beginning the tasks.
- Students need to be encouraged to actively reflect on the results of the activities as they will not do so spontaneously.
- The teacher has an important role in leading discussions and helping students focus on the process and important ideas in the inquiry and not just on the result.
- Group work is important as it fosters discussion and debate.

The tasks needed to be designed in such a way that they highlight critical aspects of the mathematical content for the students to make sense of and reason about (Olteanu, 2022). It is not enough for them to simply contain mathematics. In other words, no matter how well a task is designed to encourage discussion, reflection and reasoning, the students won't learn the desired mathematics unless the tasks have the correct mathematical focus. To this end, the tasks were designed to try and incorporate positive use of *focusing phenomenon*. Focusing phenomena, as explained by Lobato et al. (2003), are aspects of the environment, such as what the teacher says, tools and tasks, which focus students' attention on specific details. The authors show that when this is done unintentionally, it can lead to incorrect generalizations and misconceptions. However, when done carefully and intentionally, it can help lead the students to deeper mathematical insights.

This design principle was enacted by breaking down the tasks into components that specifically focused the students' attention on important mathematical concepts. For example, one task only asked the students to write a program to factor an arbitrary integer. Here the act of programming functioned as a focusing phenomenon directing the students' attention to understanding all the steps in factoring an integer, positive or negative, and determining both positive and negative factors. A follow-up task asked the students to calculate the sum of the factor pairs. Here again, isolating this aspect of the process focused their attention on critical details surrounding this procedure. The fact that they were asked to come up with these solutions themselves helped prevent detrimental focusing phenomenon by having limited extra instruction or information which might accidentally contribute to misconceptions.

The tasks were also designed to promote creative mathematical reasoning using the criteria specified by Lithner (2017, p. 939).

- Creativity: the learner creates a reasoning sequence not experienced previously, or re-creates a forgotten one.
- Plausibility: there are predictive arguments supporting the strategy choice and arguments for verification, explaining why the strategy implementation and conclusions are true or plausible.
- Anchoring: the arguments are anchored in the intrinsic mathematical properties of the components of the reasoning.

The key design principles highlighted by Lithner (2017) for encouraging creative mathematical reasoning revolve around the balance between providing students with a problem they don't know the solution to, but one for which they have a reasonable chance of constructing a solution to. Key aspects here include tasks that encourage creativity (students need to reason their way to a solution) but also embody possibility (they have the required knowledge to be able to construct a solution). In addition, the students need to take on the responsibility of constructing the solution themselves, though appropriate guidance from a teacher is still okay (as long as the teacher doesn't take over the task). All of this was encouraged by designing tasks that built on the students' previous knowledge and knowledge gained in previous tasks, while constantly adding new ideas to explore and learn.

Additional task design principles (described in Table 3 below) were borrowed from Clements (2000), who based the principles on a review of existing literature on integrating computers into mathematics education, as well as from Freudenthal et al. (2013), who developed task design principles as part of a project to teach algebra through programming. The principles from Clements (2000) include encouraging students to devise solutions, inviting students to make decisions and encouraging the students to develop theories about mathematical relationships. The principles from Freudenthal et al. (2013) include encouraging cognitive conflict, not overcomplicating tasks and encouraging the students to explore and investigate the mathematics.

Taking inspiration from the curriculum development work in Hershkowitz et al. (2002), the design process was dialectic, with research and design mutually influencing each other. As a first step, the a priori analysis as well as the literature on student difficulties with quadratic equations were used to design programming tasks for the students in the first intervention (see section 3.2). The tasks were designed to help students construct an understanding of the zero product property and its connection to solving quadratic equations (e.g., task 3, intervention 1, see section A1.1.4). The tasks were also designed to help students understand the connection between the solutions to a quadratic equation and the coefficients of the equation in standard form as well as the connection between the solutions and factored form (e.g., task 5, intervention 1, see section A1.1.6). The tasks were optimized for quadratic equations with two integer solutions with distinct absolute values, although this was not intended to prevent students from taking account of special cases, such as when one of the coefficients in standard form is 0 or when there are two identical solutions.

Table 3 provides a summary of the task design principles as well as a description of how they were operationalized.

In addition to focusing on the specific difficulties with quadratic equations found in the literature, the tasks were also designed in a way that was thought likely to encourage the students to develop their proceptual understanding of the desired knowledge elements (see section 7.1).

While the tasks were designed to allow the computer to do the calculations and test a large number of cases at the same time, the students had to specify which calculations should be performed and how to interpret the results, some of which may be used in further calculations. This was intended to help them develop a conceptual understanding of equations as objects while at the same time not losing sight of the procedures needed for working with them.

The tasks were designed to work in almost any modern text-based programming environment. Even if the examples provided in this text are written in Python, that is not to be taken as a suggestion that Python is required, or even preferred. For example, the students in the first

intervention used C++ since they were more familiar with that language.

After each intervention, the tasks were modified based on experience from the intervention as well as discussions with more experienced colleagues in research seminars. Some of the changes were of a practical nature, and some were attempts to make the tasks more in line with the design principles discussed in this section. See sections A1.2.1 and A1.3 for more information on changes made between interventions.

The tasks used in the interventions are provide in *Appendix 1 Programming Tasks*.

Table 3

Task Design Principles and Their Operationalization

Design Principle	Operationalization
Design principles connected to student engagement in the tasks	
Students should predict and discuss solution methods (Hershkowitz et al., 2002; Lithner, 2017).	The students were given example tasks with pen and paper and were asked to try and explain exactly how they completed the task. For example, before writing a program to factor an arbitrary integer, they were first given a few specific integers to factor by hand (e.g., task 4, intervention 1, see section A1.1.5).
Students should explore and investigate the underlying mathematics, creating new (for them) reasoning sequences (Clements, 2000; Freudenthal et al., 2013; Lithner, 2017).	All of the pen and paper tasks encouraged the students to explore and investigate by encouraging them to ask questions about what they were doing and by asking them to look for any patterns. Even many of the programming tasks included elements of exploration and investigation as the students were encouraged to test their results as they went to better understand the problem they were working on. Most of the tasks use this principle when trying to encourage students to notice patterns in what they are working with. The ultimate example of this being the realization of the simple connection between the solutions and the coefficients of a quadratic equation in standard form. Most of the tasks dealt with new mathematical ideas for the students in one form or another and none of the tasks were of a routine nature. This was intended to promote creativity among the students.

Students should anchor their arguments in the intrinsic mathematical properties of their reasoning and develop theories about how the underlying mathematics works (Clements, 2000; Lithner, 2017).	In order to complete the programming tasks, the students had to anchor their reasoning firmly in the relevant mathematics, otherwise the program would not produce the correct results. Most of the tasks focused on having the students develop their own ideas and theories about important mathematical relationships. The most important example here would be the relationship between the solutions to a quadratic equation and the coefficients of the equation in standard form.
Design principles connected to inherent properties of the tasks	
Tasks should lead to a cognitive conflict (Freudenthal et al., 2013).	An example of this can be seen in task 3, intervention 1 (section A1.1.4), where students were asked to write a quadratic equation from the solutions. The cognitive conflict comes up when trying to understand that each factor is x minus the solution. The negative signs cause confusion here. Another example of this can be seen in task 4, intervention 3 (section A1.3.7), where the students would start to see why there are exactly two solutions to a quadratic equation.
Tasks should build students' intuition (Freudenthal et al., 2013).	An example of this can be seen in task 5, intervention 3 (section A1.3.8), where the students had to multiply two binomials. This helped them practice their algebra skills as well as giving them a chance to reflect on how they were combining the linear terms. This worked towards building their intuition and understanding of the connection between the solutions and the coefficients of a quadratic equation in standard form.
Tasks should lead the students to make decisions and devise solutions based on plausible arguments (Clements, 2000; Lithner, 2017).	Since few of the tasks were routine problems for the students, they were constantly encouraged to make decisions and devise solutions. The tasks were built on assumed previous knowledge and on knowledge gained in previous tasks, enabling the students to make plausible arguments for the solutions to each task.
Tasks should not be overcomplicated (Freudenthal et al., 2013).	This was a difficult design principle to enact due to the difficulties the students had with programming. Part of the main evolution of the tasks from the first to the last intervention was connected to continuing efforts to enact this design principle. This can perhaps be seen most clearly in the tasks for the third intervention where most of the difficult programming elements were separated out for focused, step-by-step practice and where the students were supplied with additional code examples to use when completing other tasks (section A1.3).

Design principles connected to the environment	
Students should work in groups and engage in open discussions (Clements, 2000; Hershkowitz et al., 2002).	The students worked in groups with two or three students in each group.
Design principles connected to the teacher's role	
The teacher should direct discussions and help students focus on the process, ideas and inquiry and not just on the results (Hershkowitz et al., 2002).	Acting as the teacher in all the interventions, I gave feedback and asked questions to help the students focus on the important ideas in each task. After the first intervention, I realized the need to make this process more explicit and so developed more detailed guidelines for the teacher before the second intervention.
The teacher should encourage students to actively reflect on the results of the tasks (Hershkowitz et al., 2002).	This was done by asking them guiding questions such as: "Is there a systematic way to find all the factor pairs? What would be different if the number was -12?" (from section A1.1.5).

9.4 Ethical Considerations

Before discussing the Act concerning the Ethical Review of Research involving Humans (SFS 2003:460), which will be referred to as the Ethical Review Act from now on, it seems relevant to first discuss Karlstad University's internal review system.

All research projects at Karlstad University, including doctoral projects, must first go through the university's own internal ethical review system. The first step in this system is for the researcher to send in the relevant documents and have a case worker review them. More complicated projects will be sent to the internal Ethical Review Committee for review, whereas simpler cases can be approved by the case worker. Projects that don't need to be sent to the national ethics review board can be approved by the university's internal committee or by a case worker. All of my interventions were approved by a case worker and it was deemed unnecessary to submit the project for review by the national ethics review board. My project was assigned the internal case number HNT 2022/137,

9.4.1 Personal Data

A limited amount of personal data was collected from the students for administrative purposes and to identify possible confounding variables that may affect the results. This data was limited to each student's age, gender, current mathematics course, school and most recent grade in mathematics. The students' names were saved temporarily in case they wished to withdraw from the project but have now been deleted.

9.4.2 The Ethical Review Act

The Ethical Review Act specifies criteria for when a research project needs to be submitted to the ethical review board for approval. Most of the criteria, such those regulating physically invasive procedures or the collection of biological material, are quite unrelated to my project and not a cause for concern.

The only criteria that might require a closer look is described in paragraph 3.1 and states that any research that deals with sensitive personal data, as defined in article 9 of the EU general data protection regulation (Regulation EU 2016/679), needs to be submitted for an ethical review.

Article 9 lists the following as sensitive personal data:

> ... data revealing racial or ethnic origin, political opinions, religious or philosophical beliefs, or trade union membership, and the processing of genetic data, biometric data for the purpose of uniquely identifying a natural person, data concerning health or data concerning a natural person's sex life or sexual orientation ...

Since my project will not process any of the types of data listed in article 9, it will not be necessary to submit it for approval by the ethical review board and it will suffice with the approval of Karlstad University's internal ethical review system.

9.4.3 Information and Consent

The Ethical Review Act also specifies requirements concerning what information should be given to research participants in advance, as well as when and how their informed consent should be secured, which is also in line with the code of conduct specified in ALLEA (2017). Research subjects should be given the following information before deciding if they want to participate in the study:

- A description of the research project and what it is for.
- The purpose of the research and its methods.
- Any potential risks to participants in the study.
- Contact information for the person responsible for the research.
- That participation in the study is completely voluntary.

This information was provided to research subjects for my project through a letter of information that they read before choosing whether or not they wanted to participate. The letter was approved by Karlstad University's internal review system before being sent out.

Once the participants reviewed the information, they were asked to give their written consent to participate. Allowing them to review the information about the study ahead of time helped insure that their consent was informed consent. As all potential participants were over the age of 15, it was not necessary to obtain consent from their legal guardians (Vetenskapsrådet, 2017). Although the participants may withdraw their consent and quit the project at any time, paragraph 19 of the Ethical Review Act states that any data already collected may be kept and used in the research.

9.4.4 Anonymity

All reasonable efforts have been made to protect the identity of the students participating in the interventions. Their names do not appear in the transcribed audio at any point, and all Swedish comments have been translated into English which adds a further layer of protection. No identifying information about their schools is included in this text.

However, complete anonymity, as defined by Vetenskapsrådet (2017), cannot be guaranteed and has not been promised to the students. The audio recordings are raw data that must be saved and there is always the risk of someone being able to identify the students from the recordings. In addition, there is no way to prevent participants in the study from publicly stating that they participated in the study, which would not only allow themselves to be identified, but could lead to the identification of the other students as well. Because of this, as recommended by Vetenskapsrådet (2017), the information provided to the students only described the steps that would be taken to protect their identity without making any promises.

10 How to Read the AiC Analyses

The data from the interventions was analyzed using the RBC+C model as described above. This a posteriori analysis was used to attempt to identify the construction of the target mathematical constructs as a result of the tasks as well as to connect them to the act of programming.

Each intervention is presented in a separate section below. The analysis is presented at the end of each section and starts with a list of the relevant constructs which could be identified.

The indices for each construct are consistent between the interventions and are connected to their content in order to make them easier to recognize and remember. For example, C_0 is used to refer to constructs connected to the zero product property. The same subscript is used to indicate which construct is recognized or built with. For example, the epistemic action R_0 refers to recognizing the construct C_0 which is an understanding of the zero product property.

New constructs are formed by first recognizing and building with previous constructs. However, incomplete constructs may be recognized and built-with in order to strengthen them and move them towards consolidation. In these cases, I have treated the new and deeper understanding as a new construct, but kept the index the same for the actions of recognizing and building with previous versions of the construct. For example, R_0 refers to recognizing previous constructs connected to understanding the zero product property, but they appear under C_0, which represents newer and deeper constructions connected to the zero product property.

The students in each intervention are identified as S1, S2, etc. Within a given intervention group, S1 always refers to the same student, but refers to different students in different intervention groups.

Detailed descriptions of the tasks used in each intervention are included in *Appendix 1: Programming Tasks*.

Tables of the coded utterances for each intervention are included in *Appendix 2: Coded Transcription Lines*.

Note that the utterances from the students in Swedish have been translated to English. All translations are my own.

10.1 Additional Theoretical Considerations

In addition to the RBC+C analysis as described in the AiC framework, results were also analyzed for evidence of creative mathematical reasoning as described by Lithner (2008) as well as evidence for the development of proceptual thinking as described by Gray and Tall (1994). Connections were also made, where possible, to the three layers of algebraic thinking introduced by Radford (2006).

11 Intervention 1

The first intervention was conducted with two Swedish 9th graders. Both students were experienced programmers and were used to working together on programming projects in C++ outside of school. Both students were expected to receive an A as their final grade in mathematics.

The students needed little encouragement to participate in the study and thought it was interesting and fun as well as an opportunity to learn something new. Both students were calm and focused and did not exhibit any difficulty or unwillingness to follow instructions.

The students chose to write their programs in C++ as that was the language they were most familiar with.

The students met with me three times after school for the lessons. Each lesson lasted approximately one hour. Two of the lessons were in the same week, but due to one of the students becoming ill, the third lesson was delayed and couldn't be conducted until two weeks later.

There were 6 tasks prepared for the students, and they completed two tasks each lesson on schedule. I chose to skip over some of the pen and paper activities after deciding that they didn't gain as much from them as I had thought they would due to their programming proficiency. Having them move faster and more directly to the issues relating to programming proved to be much more productive.

11.1 Data Collection in Intervention 1

During the first intervention, the students used a computer I had provided which allowed me to install the licensed screen recording software Screencast-O-Matic ahead of time. This software had the additional advantage of recording audio as well as what was happening on the computer screen, which proved to be helpful in both matching dialogue up with concurrent programming actions, as well as with deciphering some of the harder to understand utterances. Only one voice recorder was used, placed on the desk between the students.

Both the voice recorder and the screen recording software worked as intended and good data was collected. However, an unexpected

problem occurred repeatedly during transcription; the two boys sounded so similar in tone of voice, way of speaking and choice of words, that it was sometimes quite difficult to tell them apart and to know who was actually speaking. Because of this, there may be some errors in who certain utterances are attributed to. However, as it is the group's construction of knowledge that is being analyzed, small errors in utterance attribution are unlucky to have any significant effect on the analysis.

The original intention was to proceed with a semi-structured interview with the students after the last lesson. However, due to time constraints and the determination that enough data had already been collected, the students were only asked to solve a couple of quadratic equations using pen and paper.

11.2 A Priori Analysis for Intervention 1

At the start of the study, the students had very little knowledge about quadratic equations, only having heard the term before, though they did have some experience solving equations of the form $x^2 = a$. Their basic algebra manipulation skills were good, though not problem free. They had also previously learned the FOIL method for multiplying two binomials. FOIL is commonly taught in English speaking countries and is a mnemonic to help remember to multiply all the terms with each other when multiplying two binomials and stands for First, Outside, Inside, and Last. The students were proficient programmers and had extensive experience with C++.

It was hoped that the students would construct the following knowledge elements after completing the programming tasks. Following the methodology example described in Dreyfus et al. (2015), each knowledge element below will also include a brief description of the clues that will be used to identify the related construction actions.

- **Overarching main goal**. The main goal for all the tasks was for the students to construct a proceptual understanding of how to solve quadratic equations by factoring and using the zero product property.
    - I will say that the students have reached the main goal when they can be given a monic quadratic equation and

with pen and paper can either find the solutions by factoring it over the integers or say with confidence that it has no integer solutions.

- **Zero product property (C_o)**. The students were expected to begin the intervention with some previously constructed knowledge elements related to the zero product property, but were not expected to have this construct consolidated. It was hoped that the tasks would move this knowledge element closer to consolidation.
    - I will say that students have constructed this knowledge element when they show an understanding that the product of numbers is zero if and only if at least one of the numbers is zero. It is assumed they will start the project with the understanding that the product of a number and zero is equal to zero, that is to say an *if* but not an *if and only if* understanding.
- **Factoring an integer (C_{fi}).** The students were expected to start the intervention with some previous constructs connected to factoring an integer. However, this knowledge element was not expected to be consolidated. By the end of the tasks, it was hoped that the students would have moved this knowledge element closer to consolidation.
    - I will say that students have constructed this knowledge element when they are able to clearly specify an exact procedure that they can use to find all factor pairs of an integer. This should include how they check for factors, when to stop checking for factors, as well as how to find negative factors and deal with factoring negative integers.
- **Two solutions (C_{2s})**. The students in the first intervention were not expected to know that there are always two solutions to a quadratic equation. It was hoped that they would construct this knowledge element during the tasks. In particular, it was hoped that they would construct an understanding that the two solutions are each individually valid solutions and to avoid the mistake seen in the literature where some students try and apply both solutions at the same time (see section 3.2).
    - I will say that students have constructed this knowledge element when they show an understanding that all

quadratic equations have two solutions, even though sometimes they are both the same number and sometimes they are not real numbers.

- **Equations and expressions as procepts (C_e).** The students were expected to have a procedural understanding of equations and expressions at the start of the tasks. It was hoped that they would construct a proceptual understanding of them by the end of the tasks, being able to see the equations and expressions as objects as well as understanding procedures for manipulating them.
    - I will say that students have constructed this knowledge element when they show fluency when dealing with algebraic expressions. This includes automatically combining the linear terms when multiplying two binomials, and thus seeing the connection between factors and the coefficients in the resultant polynomial. It also includes correctly dealing with negative numbers and understanding that a negative sign in front of a variable changes the variables sign and doesn't always make it negative. General fluency will be looked at in all cases where the students work with equations and expressions.
- **Connection between solutions and factored form (C_{ffs}).** By the end of the tasks, it was hoped that the students would construct an understanding of the connection between the solutions to a quadratic equation and that equation in factored form.
    - I will say that students have constructed this knowledge element when they show an understanding that a factorable quadratic equation can be written as $(x - r_1)(x - r_2) = 0$, where r_1 and r_2 are solutions to the equation. In particular, they need to show an understanding that the additive inverse of the solutions appears in the factors.

- **Connection between solutions and standard form (C_{st}).** By the end of the tasks, it was hoped that the students would construct an understanding of the connection between the solutions to a quadratic equation and that equation in standard form, in particular the connection between the solutions and the coefficients.
    - I will say that students have constructed this knowledge element when they have themselves discovered Vieta's formulas for quadratic equations. In other words, they understand that for a monic quadratic equation in standard form $x^2 + px + q = 0$, with solutions r_1 and r_2, that $p = -(r_1 + r_2)$ and $q = r_1 r_2$.

11.3 AiC Analysis of Intervention 1

Table 4 below lists the relevant constructs, along with their connected epistemic actions, identified in the first intervention. This includes constructs identified from the a priori analysis, as well as new constructs seen in the a posteriori analysis. The epistemic actions identified were recognizing (R), building-with (B), and constructing (C).

Table 4

Constructs Identified in Intervention 1

Label	Description and comments
R_p B_p	These epistemic actions refer to previous constructs in general where it was deemed unnecessary to specify them individually, yet they seemed to play a role in a new constructing action.
R_o B_o C_o	The zero product property. This was a previous construct the students recognized and built with during the activities. This construct was not consolidated before the lessons and further construction moving towards consolidation was seen.
R_{fi} B_{fi} C_{fi}	This construct relates to the understanding of the factors of an integer, in particular as a procept. The students started with some basic constructs, but were able to develop them more during the tasks.
R_e B_e C_e	This construct is connected to a full understanding of equations and expressions as procepts, both as something that that calculations can be performed on and with, as well as equations and expressions as objects that may represent an idea or situation. The students had previous constructs of equations and expressions, but these were built upon and further constructs were formed, moving them towards a consolidated understanding of equations and expressions.
R_{2s} B_{2s} C_{2s}	The fact that there are always two solutions to a quadratic equation. This was a new construct seen during the lessons.
R_{mb} B_{mb} C_{mb}	How to multiply two binomials. This was a previous construct. However, it was not consolidated, so further construction was seen with it during the lesson, moving it towards consolidation.
R_{ffs} B_{ffs} C_{ffs}	The connection between an equation in factored form and the solutions to the equation. This was a new construct seen in the lessons.
R_{st} B_{st} C_{st}	This construct is an understanding of quadratic equations in standard form, in particular how the coefficients relate to the solutions and to factored form. This includes factoring it when possible.

Each of the following sections highlights relevant constructing actions connected to each specific construct. Evidence is also provided and

discussed in cases where the act of programming appears to have facilitated the construction process. Example epistemic actions were chosen to exemplify the process for each particular construct. A more detailed coding of the epistemic actions for each construct can be found in Appendix 2 *Coded Transcription Lines*.

11.3.1 C_0: The Zero Product Property

The students showed some previous understanding that if the product of two numbers is zero then one of them must be zero. They were able to recognize and build with this construct. For example, from task 1:

3, S1: "You can say if number is just two and the other is zero then the product is zero."
10, S2: "Nothing else will make it zero between them."
13, S1: "Not when it's multiplication."

An example from task 2:

112, S1: "Yeah, so … no matter what we should always make it equal to zero. Because …"
113, S2: "Yes, because it will always be zero."

However, some uncertainty could be seen in some of their utterances indicating that their knowledge was not consolidated. Further construction was identified in tasks 1, 2, 3 and 6. Programming did not seem to aid in this process. For example, from task 1:

27, S1: "I don't think it's possible to do it with anything other than zero to get a zero for the product."
31, S2: "There's almost nothing else you can times it with so it will be …"

Here is another example from task 6:

668, S2: "Nothing more, right?"
669, S1: "What else can it be?"

This seemed to indicate that while the construct was there to begin with, it was not yet consolidated. In other words, the students demonstrated a basic understanding of the zero product property, but were not yet completely sure of its full scope. Another way to understand the difference is that the students displayed *if* understanding, but not *if and only if* understanding. For example, compare the following two statements:

- If one of two numbers is zero, then their product is zero.
- The product of two numbers is zero if and only if one of the numbers is zero.

The students showed confidence with the first statement, but some hesitancy with the second. However, throughout the course of working

on the different tasks their certainty increased and they appeared to grow more confident in the *if and only if* understanding. For example, from task 6:

680, S1: "It doesn't matter, it is always equal to 0."

11.3.2 C_{fi}: Factors of an Integer

The students had some previous understanding of how to factor an integer and so were able to recognize and build with this construct. However, their knowledge was not consolidated and so further construction was witnessed in the lessons. All of the epistemic actions in this section were identified during task 4 and programming seemed to aid in part of this process.

At the beginning of the task, the students appeared to recognize a previous construct connected to factoring integers.

479, S1: "You can divide six by different things. That's how we get this."

They were able to write down the factors of 6 on paper without much trouble. But when asked to write a program to factor an arbitrary integer, they suddenly ran into the limits of their construct.

485, S2: "What's the fastest way? Dividing by two is easy, you get what you can times with two. But it doesn't work for odd numbers."

Here the student appeared to be referring to simple division rules, starting with dividing by two. They looked at 6 and saw it could be divided by 2 and easily ended up with $2 \cdot 3 = 6$. This indicates that there was some idea of divisibility, but only with specific numbers. The student appeared to realize the problem quite fast, noting that that wouldn't work for odd numbers, showing the beginning stages of creative mathematical reasoning here (Lithner, 2008).

At this point, I asked the students if they knew how to programmatically test for divisibility. They knew about the modulo operator, and the question was enough to get them going and programming.

Even with the hint and a new idea of how to proceed, the students struggled with understanding exactly how to create the loop and divisibility test.

496, S1: “Wait, it shouldn’t be x there … if two times three … it’s … wait, I have to think.”

```
for (int i = 0; i < x; i++) {
  if (x % i == 0) {
  }
}
```

This indicated they were still in the process of constructing new ideas about how factoring works. The technical details of programming were not difficult for them, but understanding how exactly to use the variables in the loop as well the starting and ending value of the loop were not initially clear to them. The compiler helped them with some of these problems, such as giving them a division by zero error which helped them realize the loop should start at 1 and not 0.

The students initially thought they would have to write separate code for factoring negative integers, but quickly realized the divisibility checks would work the same as long as they fixed the upper limit of the loop.

510, S1: “That should also work if we write negative 6.”
532, S1: “So we make this positive and when we find them we make one negative and then the other.”

This indicated that the need to write a program to factor both positive and negative integers motivated them to use creative mathematical reasoning which led them to construct a deeper understanding of the connection between positive and negative integers and their factors.

11.3.3 C_e: Understanding Equations and Expressions

Although the students had previous knowledge constructs connected to the definition and use of equations and expressions, they were seen to recognize and build upon these constructs to move towards consolidating the understanding of equations and expressions as procepts, both as something that calculations can be performed on and with, as well as an object that may represent an idea or situation. Because of this, they were able to use recognizing and building-with actions for previous constructs related to understanding an equation to construct a new and deeper understanding. Some of these epistemic actions appeared to be connected to programming.

During the lessons, the students started out focusing on definitions, and were initially unsure of the difference between an expression and an equation and how they are defined.

116, S1: "It's not an equation. It's a …"
117, S2: "No, right, it's not one."
118, S1: "Or yeah, it is one."

The students demonstrated good procedural skills involving basic algebraic manipulations, but their discussions during task 2 showed they had trouble thinking of an expression as an object, inhibiting a proceptual understanding of expressions as well as indicating trouble with the symbolic layer of algebraic thinking. For example, in the expression $x(x-2)$ they understood that the variable x is the same in both factors, but had trouble understanding that the two different factors to be considered here are x and $(x-2)$, each of which is a different number itself.

121, S1: "It's pretty clear that you can write … if we take … x … times minus two, for example. But that is like one way. It works. If we enter a positive number so that x is two then this works. Right?"

Later in the task, there was evidence that they started to construct an understanding that an expression such as $(x-2)$ can also be thought of as a number itself, not just a set of symbols to be manipulated to get an "answer", thus moving the expression from procedure to procept as well as increasing their fluency with the symbolic layer of algebraic thinking.

131, S1: "Yeah. It doesn't represent it because it isn't two. It has to be a two for [*mumbles*] can't change. Then it has to be a two. But … the problem here is if x is a negative number then it doesn't work. So it has to work in some way for both positive and negative."
149, S1: "You can … I don't know … we can expand here if it's possible or not. But otherwise we can write … the same but with x plus two since it then works for negative. But then we have two which work …"

Part of this construction seemed to be motivated by the programming part of task 2, which in turn seemed motivate the students to engage in creative mathematical reasoning. The requirement to program a general solution to this task, including outputting the values for x seemed motivate further construction of an understanding of equations and expressions.

284, S1: "x can be negative four, zero, and x can be negative two and the difference is still two."

Here there were indications of a shift to proceptual thinking: from thinking of $(x - 2)$ as something to be calculated to also thinking of it as an object itself. The expression moved from procedure to procept.

299, S1: "So we have to … x can be equal to either negative … or it can be equal to … it is … or should it be equal to negative itself and zero … zero always works."

Constructing actions connected to this knowledge element were even seen during task 3.

372, S1: "But wait … I realized something I did wrong. But …"
377, S1: "The six should be like this … yeah, it's last … okay."

11.3.4 C_{2s}: Two Solutions to a Quadratic Equation

The understanding that there are always two solutions to a quadratic equation was a new construct seen during the tasks. While working with different tasks by hand, the students were able to use their knowledge connected to equations and expressions together with their understanding of the zero product property to begin to see that there are always two solutions to a quadratic equation. For example, from task 2:

67, S2: "There are two solutions every time."
68, S1: "Yeah, there are."

The tasks were designed to build up this understanding slowly in specific simplified situations, and to allow them to make the finally generalized connection to quadratic equations with integer solutions. This connection stemmed from the developing understanding, through the use of creative mathematical reasoning, that these equations could be factored, and that each factor represented a different solution. This also showed progress towards the symbolic layer of algebraic thinking. For example, from task 3:

334, S1: "Okay. Ah, yeah. So for this one it should be negative 3 and negative 4."

Progress in their understanding could be seen, but it remained uncertain at times. In each case with pen and paper they were working

on a specific problem with predetermined values. Each time a similar situation arose in the different tasks they were able to come to the right conclusions, but with some hesitancy, which seems to indicate incomplete construction of this idea and that they may have been stuck in the contextual layer of algebraic thinking, as well as perhaps a higher focus on the procedural aspects of what they were working on. For example, from task 6:

668, S2: "Nothing more, right?"
669, S1: "What else can it be?"

In the final task, they were asked to write a program to check if a given quadratic equation with integer coefficients could be factored, and if so, to factor it, derive the solutions from the factors, and to show calculations checking the solutions. It was in taking these final step with the program, when they needed to generalize the procedure in a concrete way, that they became sure of themselves and seemed to gain an understanding of how the principle generalized and thus not only develop a better proceptual understanding but also increased fluency with the symbolic layer of generalized algebraic thinking.

699, S1: "Now we've got the first answer. We have to do the same thing with the other one."
700, S2: "It's exactly the same there."

It was there as well, while programming, that the students showed a deeper understanding that each factor represented a different possible value for x, but when testing a solution, the same value had to be used for the x in each factor.

697, S1: "We have to show our calculations … we can write equal to zero here so we actually solve something."

```
printf("(%d   +   %d)(%d   %d)=0\n",   solution_first_pair,
f.first_pair, solution_first_pair, f.second_pair
```

Research into student misconceptions with quadratic equations shows that checking both answers at the same time (i.e., using two different values for each occurrence of the variable) is a common problem among students learning how to solve quadratic equations (see section 3.2). During this programming task, the students were able to avoid this common misconception. Although it is not possible to know if the students would have avoided this misconception as easily without

engaging in the programming tasks, there does seem to be evidence that the tasks at least facilitated the learning process here.

11.3.5 C_{mb}: Multiplying Two Binomials

Although the students had previous constructs connected to multiplying two binomials, they were seen to recognize and build upon these constructs to move towards consolidation. This construct overlaps quite a bit with C_e but was relevant and specific enough to warrant its own definition. This became apparent during the a posteriori analysis as this was a particularly important aspect of understanding expressions for these tasks. All of the epistemic actions in this section were identified during task 3 and none of them were connected to the act of programming.

The students started the intervention with previous knowledge of a procedural trick for multiplying two binomials known as FOIL, which stands for Frist, Outside, Inside and then Last. This procedural mnemonic is commonly taught in a number of English speaking countries. However, this procedural knowledge seemed to be relatively insecure, which caused the students problems on several occasions.

360, S1: "But you just need to use the distributive property? I guess. Is that right?"
365, S1: "But I don't think so … but then … it would be more x's than it would be here."

One of the mistakes they ran into was failing to combine the two linear terms, thus ending up with four terms instead of three. This seemed to stem from a heavy reliance on an uncertain procedure. Their focus was on completing the steps in FOIL, and they did not seem have a proceptual understanding of the expression at this point.

368, S1: "That's not right. But … I think … no. But … if we could start with something like this and then see how they got here then it would have been much easier."
382, S2: "I don't know. This isn't right. But this is right. So if you just … FOIL … expand out then it becomes standard form since this is standard form."

Their focus on procedure here seemed to hamper their ability to engage in creative mathematical reasoning and seems to show them having trouble breaking out of the contextual layer of algebraic thinking.

11.3.6 C_{ffs}: Connection Between Factored Form and Solutions

As all of the tasks were designed to try and help the students construct an understanding of the connection between the solutions to a quadratic equation and that equation in standard form, it is not surprising that epistemic actions connected to this construction process can be seen throughout the lessons. The need to write a program to solve various tasks seems to have helped the students construct this knowledge element at different stages.

Initial construction actions were seen during task 3 where the students were first asked to write down a quadratic equation in factored form that has the given integer solutions. They were able to do this with pen and paper but there was some degree of uncertainty in their process. In particular, they seemed stuck in the factual layer of algebraic thinking since they were still focused on finding the specific equation for the solutions given, and were not focusing on general principles.

334, S1: "Okay. Ah, yeah. So for this one it should be negative three and negative four."

However, later in task 3, they were working on their program to generate a quadratic equation for any given integer solutions. It is during this process that through the use of creative mathematical reasoning they were able to transition to the symbolic layer of algebraic thinking which suddenly made the process much simpler for them.

409, S2: "Factored is really easy, it's just changing …"
414, S2: "This should be … if they enter something positive it should be negative …"
418, S2: "If x is less than zero it should be … otherwise it should be … no! It will always be like that!"

During task 4, the students had to write a program to factor an arbitrary integer and the output included the sum of each factor pair. Writing a program to calculate the sum each time seemed focused their attention on this process.

523, S2: "Now we get the right sum anyways."

In the example above, they were fixing some small bugs in the program, which in itself required paying extra attention to the output and how it is calculated. This seemed to have made this knowledge more readily

accessible for when they later engaged in creative mathematical reasoning and put the pieces together, connecting the sum of the integers to the coefficient of the linear term.

The next important round of construction here seemed to happen during task 5 which required them to write a program to try and factor an arbitrary quadratic equation with integer coefficients. In the lines below, the students were working on a program that tries to factor the given equation by first factoring the constant term and then checking the sum of the factors against the additive inverse of the linear term, showing increased fluency with the symbolic layer of algebraic thinking.

603, S2: "Then it should check c, right?"
606, S2: "And if there's a sum that's b."

Although this is an integral aspect of the next construct as well, the creative mathematical reasoning used here seemed to have played a part in this construct as it was a necessary step to identify the solutions and to produce the factors. The act of programming seemed to help them gain a deeper understanding of the relationship between the coefficients and the factors of the constant term.

In task 6, the students put it all together, including handling special cases such as when the constant term or the coefficient of the linear term are equal to zero.

783, S2: "If c is equal to zero …"
784, S1: "Then we just write …"
787, S1: "And it's b here."

The need to write a program that worked on a general method, output both factored and standard form, output the solutions, and showed calculations proving that the solutions were correct, seemed to be instrumental in helping the students construct this knowledge element through the use of creative mathematical reasoning. The deeper understanding of the relationship between the coefficients, the solutions and the factors, seemed to be a more proceptual understanding of these equations compared to a purely procedural one. In addition, the students seemed to show even more fluency with the symbolic layer of algebraic thinking during this process.

The act of programming seemed to also have created a stronger motivation to deal with the special cases, such as when one of the coefficients are zero. This was motivated by first noticing that they did not appear in the output, and then trying to figure out why by analyzing the program, and then engaging in creative mathematical reasoning about the limitations of a given process.

11.3.7 C_{st}: Connection Between Standard Form and Solutions

Even though this knowledge construct overlaps quite a bit with the previous one, it seemed relevant to treat them separately given their individual importance in the process. However, many of these epistemic actions seemed to happen in parallel with the previous ones, possibly due to the nature of the tasks. Note that this construct captures all of the most essential elements of the desired learning outcomes, and so in that sense could be considered the most relevant one in this study.

The first noticeable contribution from programming to this construct appeared during task 3 when they had to generate quadratic equations from solutions, in both factored form and in standard form. They were quite quick to understand the connection between the solutions and factored form. With pen and paper, they first wrote down factored form, and then used the FOIL method to multiply the two binomials to find the standard form.

Initially, with both pen and paper and when programming, they had trouble realizing they should combine the linear terms, thus ending up with four terms instead of three, showing trouble moving out of the contextual layer of algebraic thinking.

With pen and paper they were eventually able to make the connection to their algebra manipulation skills, moving from a contextual to a symbolic layer of algebraic thinking, as well as moving towards a more proceptual understanding of the equation. However, when initially implementing it in their program, they tried to use the FOIL method, and were forced to consider programmatically how to combine the terms.

453, S1: “These two numbers which we calculate here have to be reused as … or, to get it in standard form … because it’s not the same as here even

if it looks like it. Or we can just ignore them because they are just negative and just write here …"

```
printf("x^2 + %dx +%dx + %d\n",fy,fx,fy*fx);
```

The need to write a general procedure in a program seemed to motivate them engage in creative mathematical reasoning in order to bypass FOIL and thus move from a meaningless procedure to an algorithm based on a proceptual understanding of how all the pieces related to each other as well as moving to the symbolic layer of algebraic thinking.

458, S1: "Right, it should also actually put those two together."

```
printf("x^2 + %dx + %d\n",fx+fy,fy*fx);
```

As it was not entirely easy to follow their partially verbalized dialogue here, I asked some follow-up questions to give them a chance to explain their reasoning here, which matched a correct understanding of the connection between the solutions and the coefficients.

470, Me: "How did you get coefficient for the middle term?"
471, S2: "It's just adding them together."
472, Me: "And the last term?"
473, S2: "We multiply them with each other."

The rest of the construction was seen during task 6. As described in the previous section, the act of putting all the pieces together in a program to try and factor an arbitrary quadratic equation with integer coefficients, finding the factors, finding the solutions, and then outputting it all, including calculations to verify the solutions, seems to have helped further the construction of this knowledge element. The following is some example output they were testing during task 6:

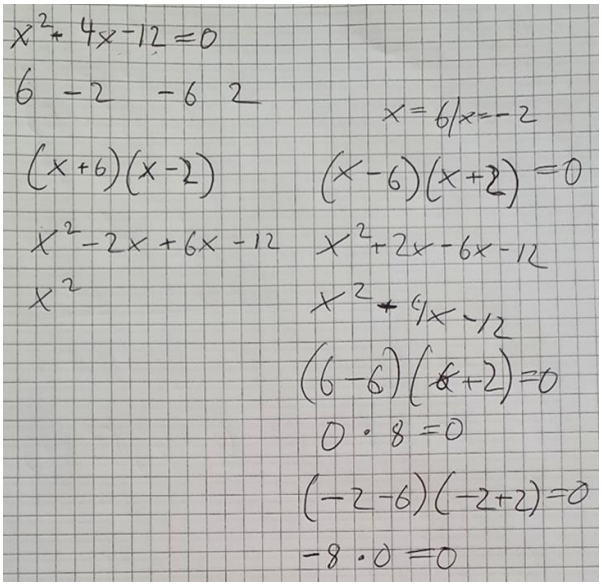
```
x^2 + -5x + 4 = 0
(x + -1)(x + -4)
(1 + -1)(1 + -4) = 0
0 * -3 = 0
(4 + -1)(4+ -4) = 0
3 * 0 = 0
```

Here they showed the correct factorization of the equation in standard form. They also showed they were able to test the solutions separately, correctly substituting for x and calculating the new factors, verifying their answer. All of this was done programmatically, showing an understanding of the general principles involved.

As a final step to try and determine if the knowledge they gained from the programming tasks would help them solve similar problems with pen and paper, the students were given two equations to try to solve by factoring. One was in standard form, while the other required an extra step to change it to standard form. They had a little bit of trouble initially with the second one, but had very little trouble factoring and solving them from standard form (aside from an initial sign error in the first one). Figure 4 and Figure 5 show their solutions.

Figure 4

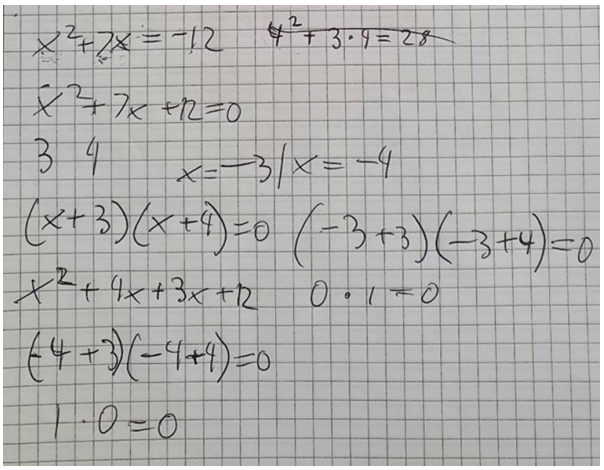
Students Solve $x^2 - 4x - 12 = 0$ *With Pen and Paper*



844, S1: "Six and two! That's it! Six and two."
874, S1: "Okay, okay. Negative six and then …"
875, S2: "Positive two …"

Figure 5

Students Solve $x^2 + 7x = -12$ With Pen and Paper



933, S1: "But then … if you could rearrange so in some way … in some way it's equal to zero. Or change it so its equal to zero …"
941, S1: "Okay, it's something which times to 12 but adds to seven."
953, S1: "Okay, then the answer is negative three and negative four."

11.4 Summary of Results From Intervention 1

Results from the first intervention provide evidence that programming tasks can help students learn to use the zero product property to solve quadratic equations. At a number of points throughout the tasks, the need to program appeared to motivate the students to engage in creative mathematical reasoning which helped them transition from a procedural understanding to a proceptual understanding of key concepts.

An example of this process was seen when they needed to write a program to factor an arbitrary integer, and important step in the overall process. The conceptual progress the students made here can be looked at in terms of the three layers of generality within algebraic thinking as described by Radford (2006): *factual*, *contextual* and *symbolic*. Although at first glance, factoring an integer may not seem to require more than factual algebraic thinking, or at most contextual algebraic thinking, the requirements of writing a program to factor an arbitrary integer seemed to require the students to develop their symbolic algebraic thinking. In order to write the program, they needed to be able to fully represent the problem symbolically and to understand the

meaning of those symbols, as well as being able to create general rules for manipulating those symbols.

When the students first started working on the factorization problem with pen and paper, they displayed factual algebraic thinking; they could perform the process with specific numbers, but had trouble describing it verbally. After they were given some help, they were able to start to describe the process verbally, though still in a contextually dependent manner. It was only when they had to implement the generalized procedure in a computer program that they seemed to finally take the step to partially freeing the symbolic representations from their context and thus exhibit symbolic algebraic thinking. Note, however, that due to the nature of the task, it would be difficult to entirely separate the context and procedure from the symbolic representations, though this is not problematic when seen through the lens of gaining a proceptual understanding of integer factorization.

This was also seen when the students transitioned from a procedural understanding of how to multiply two binomials using FOIL, to a proceptual understanding of a quadratic equation in both factored form and standard form, in particular connecting the solutions with the coefficients in standard form.

The need to write a program that worked on a general method to try and factor an arbitrary quadratic equation with integer coefficients, finding the factors, finding the solutions, and then outputting it all, including calculations to verify the solutions, seems to have been instrumental in motivating the students to engage in creative mathematical reasoning in order to construct the knowledge necessary to solve a quadratic equation by factoring and using the zero product property. Direct application of this construct was seen in the final task where they quickly and successfully solved a pair of quadratic equations using this method with pen and paper.

Programming tasks that required the students to programmatically verify both solutions seems to have been instrumental in helping them avoid the problem seen in literature where some students mistakenly check both solutions at the same time.

Programming tasks that required them to programmatically convert from factored form to standard form, and vice versa, and both finding solutions and creating equations from solutions, all seemed to contribute to the students understanding the structure of a quadratic equation, as well as using factoring and the zero product property.

12 Intervention 2

Intervention 2 took place at a Swedish upper secondary school with four students in the first year of a technology program. All four students were 16 years old and had received a B or higher as their most recent mathematics grade. The students were currently studying Mathematics 1C.

The students met with me three times within the same week during their normal mathematics lesson times. The first two lessons were 90 minutes long and the last one was two hours long.

One of the students dropped out of the study after the first lesson and was replaced by another volunteer who participated in lessons two and three. The students worked in groups of two, though one student switched groups after the first lesson replacing the student who dropped out.

The students brought their own computers and used them to complete the tasks. They wrote their code in Python in Visual Studio which was already installed on their computers for use in their own programming course. During the second lesson, the new student brought an additional computer and kept writing code on it in parallel with the other student instead of cooperating with them to solve the tasks together on one computer. This was done in spite of my requests to only work on one computer and to discuss the tasks together.

Although their teacher had said that the students were friends and spent their free time together, some of them seemed to have difficulties cooperating and working together. The student who switched groups in the second lesson was quite social and worked well with their partner and for the last two lessons they were able to have good discussions and cooperation. However, the other group showed minimal ability to cooperate. In addition, both the students in this second group showed minimal interest in participating in the project, which created practical problems as well putting me in a complicated situation ethically.

None of the students were particularly proficient at programming and had trouble with basic concepts such as `for` loops and how to use variables effectively.

The students' lack of programming experience made it necessary to spend more time than in the first intervention on the pen and paper tasks at the start of some of the lessons. It was also necessary to have the students return to pen and paper at various times during the programming tasks to help them explain their methods more explicitly to me.

12.1 A priori Analysis for Intervention 2

At the start of the study, the students had very little knowledge about quadratic equations, only having heard the term before, though they did have some experience solving equations of the form $x^2 = a$. Their basic algebra manipulation skills were good, though not without problems, and they knew how to multiply two binomials. In addition, they had all had almost a full term of a programming course where they were learning to program in Python using Visual Studio.

The students were expected to construct the same knowledge elements after completing the programming tasks as described for intervention 1 (see section 11.2).

12.2 Data Collection in Intervention 2

During the first two lessons of the second intervention, both groups used their own computers. Since I was not able to install the licensed screen recording software on their computers, I had the teacher ask them to install free screen recording software called Free Cam ahead of time. The students who came to the first lesson all came with the software installed but not tested. I hadn't tested the software much myself ahead of time and so did not foresee some technical complications that arose.

The new student who came for the last two lessons to replace the one who dropped out brought his own computer but did not have screen recording software installed. He also ignored my requests to cooperate with his partner and only write code on the computer running the software. Because of this, I have no recording of any of the work he did. Furthermore, he communicated very little with his partner, so I had an additional problem obtaining usable data via the voice recorder. I tried to ameliorate this problem by asking them questions to try and provoke useful responses but met with limited success in this regard.

I did not realize it at the time, but the freeware did not record audio at any point, even though I thought it had a setting to do so. Unfortunately, this contributed to difficulties when analyzing the data.

To help solve some of these issues once I identified them, I gave the more active group a computer on which I had installed the licensed version of Screencast-O-Matic. This worked much better since the files were automatically saved on my own hardware when the lesson was over. This software also recorded audio from the group.

Another unforeseen practical problem further contributed to difficulties in analyzing the data. In order to try and foster inter-group cooperation, I had seated the two groups across from each other. Unfortunately, the voice recording devices turned out to be more sensitive than expected and during times of parallel conversations the recordings are a bit too chaotic to transcribe accurately.

12.3 AiC Analysis of Intervention 2

Due to the poor quality of the data obtained during intervention 2, an AiC analysis was not performed.

13 Intervention 3

Intervention 3 took place at a different Swedish upper secondary school with five 16-year-old students, all of whom had received an A as their latest grade in mathematics. Three of the students were in the first year of a natural science program and the other two students were in the first year of a technology program. The students were currently studying Mathematics 1C.

The students worked in two groups, a group of two (referred to as group 1 here) and a group of three (referred to as group 2 here). All five were calm, focused and followed directions well. Even when they thought the tasks were difficult, they appeared motivated to find a solution.

The students met with me three times within the same week during their normal mathematics lesson times. The first and last lessons were 80 minutes long, and the second lesson was 60 minutes long. The first and last lesson were allowed to run up to 95 minutes and the second lesson was allowed to run up to 90 minutes.

The fifth student had some previous programming experience in Java, and appeared to have more experience with computers in general. This proved to be quite helpful during the intervention as this student used their own computer in the group of three and was able to install additional screen recording software without trouble.

13.1 A Priori Analysis for Intervention 3

At the start of the study, the students had very little knowledge about quadratic equations, however their teacher had covered some of the basic ideas in the weeks prior to the intervention to help prepare them. This included a basic overview of a quadratic equation in standard form, graphing some simple cases and how to identify the solutions on the graph, and solving some basic equations of the form $ax^2 + c = 0$.

Their basic algebra manipulation skills were good, though not problem free, and they knew how to multiply two binomials. Four of the students had very limited experience with programming, all of which came from a few short lessons during their normal mathematics class. They had worked a little with programming in Excel sheets followed by

a little bit of Python programming. The fifth students, however, had some previous programming experience with Java.

In order to give them a little more programming experience before the intervention, as well as potentially saving some time, I suggested the teacher could try and run the first task with the entire class. This was done the week before the intervention. The teacher indicated that the students had fun with the tasks though only one group was able to hand in a solution in the time allotted to them.

The students were expected to construct the same knowledge elements after completing the programming tasks as described for intervention 1 (see section 11.2).

13.2 Data Collection in Intervention 3

During the third intervention, the group of two students used a computer I had provided with Screencast-O-Matic already installed on it. The group of three students used a computer brought by one of them which had the freeware OBS installed on it. Screencast-O-Matic recorded audio as well, however OBS did not. In addition, there was an undetected problem with the OBS recordings, causing the video to freeze and jump, which limited the access to video information for this group.

All of the students wrote their programs on *replit.com*, a well-known online programming environment. This also made it quite simple for me to provide example code for the students in the same environment.

I also collected all work the students did with pen and paper.

In order to avoid some of the problems from the second intervention, the two groups were seated quite far apart, which resulted in clear and unambiguous voice recordings.

13.3 AiC Analysis of Intervention 3 Group 1

Table 5 below lists the relevant constructs, along with their connected epistemic actions, identified in the this intervention. This includes constructs identified from the a priori analysis, as well as new constructs seen in the a posteriori analysis. The epistemic actions

identified were recognizing (R), building-with (B), and constructing (C).

Table 5

Constructs Identified in Intervention 3 Group 1

Label	Description and comments
R_p B_p	These epistemic actions refer to previous constructs in general where it was deemed unnecessary to specify them individually, yet they seemed to play a role in a new constructing action.
R_o B_o	The zero product property. Construction of this knowledge element was not identified during the lessons as the students seemed to already have a good understanding of the zero product property. However, they did recognize and build with this knowledge element while constructing other knowledge elements.
R_{fi} B_{fi} C_{fi}	This construct relates to the understanding of the factors of an integer, in particular as a procept. The students started with some basic constructs, but were able to develop them more during the tasks.
R_e B_e C_e	This construct is connected to a full understanding of equations and expressions as procepts, both as something that that calculations can be performed on and with, as well as an equation as an object that may represent an idea or situation. Construction of this knowledge element was not identified during the lessons as the students seemed to already have a good understanding of equations and expressions. However, they did recognize and build with this knowledge element while constructing other knowledge elements
R_{mb} B_{mb} C_{mb}	How to multiply two binomials. This was a previous construct. However, it was not consolidated, so further construction was seen with it during the lesson, moving it towards consolidation.
R_{ffs} B_{ffs} C_{ffs}	The connection between an equation in factored form and the solutions to the equation. This was a new construct seen in the lessons.
R_{st} B_{st} C_{st}	This construct is an understanding of quadratic equations in standard form, in particular how the coefficients relate to the solutions and to factored form. This includes factoring it when possible. This was a new construct seen in the lessons.

Each of the following sections highlights relevant constructing actions connected to each specific construct. Evidence is also provided and discussed in cases where the act of programming appears to have facilitated the construction process. Example epistemic actions were

chosen to exemplify the process for each particular construct. A more detailed coding of the epistemic actions for each construct can be found in Appendix 2 *Coded Transcription Lines*.

13.3.1 C_{fi}: Factors of an Integer

The students had worked on the task connected to factoring an integer the week before the intervention and demonstrated some previously constructed knowledge here. However, their knowledge was not consolidated and so further construction was witnessed in the lessons. Programming seemed to aid in part of this process.

When working with pen and paper, they showed proficiency in finding the factors and even some recognition of when they should stop checking. From task 1 on paper:

6, S1: "One times six, two times three. But this is what we said. The square root of six is …"
10, S1: "Four isn't included. Six times one and then … so it's … exactly, that's it."
15, S1: "If it had been minus six … we used like the same thing. Minus one times six. Minus two times three. Minus three times two. Minus six times one. Yes."

They seemed to be able to find the factors without fully understanding the process they were using and had trouble articulating exactly why they did what they did indicating that they were perhaps working in the contextual layer of algebraic thinking. From task 1 on paper:

30, S1: "Twelve … four times eight. Five doesn't work. Six works, right?"
33, Me: "Good. Now you are saying *it works*, *it doesn't work*. What do you mean?"
34, S1: "That the number can't be multiplied by another even number."

When they started trying to write a program to factor an arbitrary integer, instability in their understanding started causing them trouble and forced them to consider their process in more detail. From task 1 on the computer:

51, S2: "We can do that, but it will be quite hard since we don't know how far to go. We need to go to the square root of the number. Or the one under it. We don't know which number that is."

This example shows the beginning of some creative mathematical reasoning. The students start to construct an argument that they only need to loop up to the square root of the number to check for factors.

However, they had trouble going from the specific case to the general case with their code. From task 1 on the computer:

146, S1: “No, we want it to calculate it.”

```
print((tal/2),"2")
```

Unfortunately, their lack of programming experience prevented them from making any real progress on this task and although there is some evidence for the need to program motivating further construction of this knowledge element, there is no good evidence that it happened here.

13.3.2 C_e: Understanding Equations and Expressions

Although the students had fairly well constructed knowledge pertaining to equations and expressions, some of the programming tasks highlighted some insecurities in their knowledge. In the following example from task 5 on the computer they were trying to figure out how to calculate the coefficient of the linear term, which should be the negative of the sum of the roots. However, they were struggling with the minus sign.

645, S1: "No, it should be minus x."
646, S2: "Like this."

```
(r1-r2,r1*r2)
```

655, S1: "Because that's positive and that's negative. So maybe we should write minus r1?"

```
(-r1+r2, r1*r2)
```

659, S1: "It's always negative, so we can have parentheses around it, right?"

```
(-(r1+r2), r1*r2)
```

These examples show the beginning of creative mathematical reasoning, however their reasoning in not yet entirely correct. Even though the last bit of code above is technically correct, their motivation isn't entirely valid.

During task 6 on the computer they showed further construction towards generalizing the use of the minus sign.

749, S1: "It depends on if the number they say is positive or negative. Or yeah, it should be changed, so it has to be minus since that changes it."

Here they finally make full use of creative mathematical reasoning, anchoring their logic correctly in mathematics. In Python, placing a negative sign in front of a variable changes the sign of the variable's value and doesn't just make it negative, just like it does in mathematics. In this task, the need to program a general method seems to have motivated their construction of mathematical knowledge.

The insight into this import use of the minus sign can also be seen as an example of the students moving towards the symbolic layer of generalized algebraic thinking.

13.3.3 C_{mb}: Multiplying Two Binomials

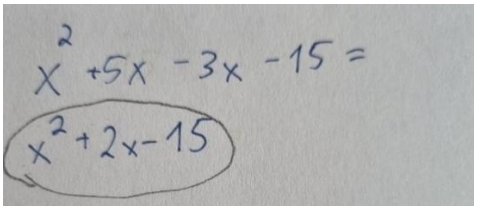
Although the students had previous constructs connected to multiplying two binomials, they were seen to recognize and build upon these constructs to move towards consolidation during task 5 on paper and on the computer. This construct overlaps quite a bit with C_e but was relevant and specific enough to warrant its own definition. None of these epistemic actions seemed connected to programming.

Like other groups, they initially had trouble automatically combining the two linear terms and immediately seeing the relationship between the binomials and the final coefficients.

491, S2: "We don't have a rule we can follow here … minus 15 … and then minus 3x."

Figure 6

Students Learn to Combine the Linear Terms

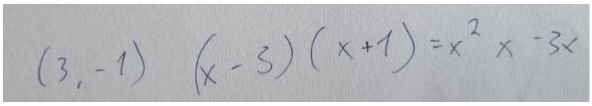


While checking their program's output, they seemed to have gained more proficiency with working with the product of two binomials, as seen in Figure 6 and Figure 7, though there is no obvious way to attribute this further construction to the act of programming.

666, S1: "Yes … and then minus 3. That works."

Figure 7

Students Check Part of Their Program's Output



13.3.4 C_{ffs}: Connection Between Factored Form and Solutions

As all of the tasks were designed to try and help the students construct an understanding of the connection between the solutions to a quadratic equation and that equation in factored form, it is not surprising that epistemic actions connected to this construction

process can be seen throughout the lessons. The need to write a program to solve various tasks seems to have helped the students construct this knowledge element at different stages. At least to some degree.

In task 3 on the computer they started to recognize the zero product property as part of their path towards constructing this knowledge element.

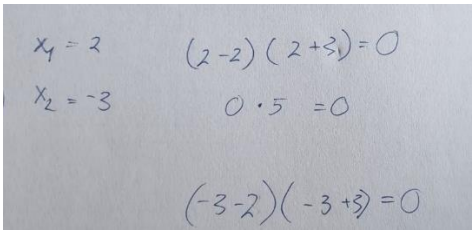
254, S1: "It makes sense that it's zero since once of the parentheses is zero."

Initial construction actions could already be identified in task 4 on paper (see Figure 8).

269, S2: "One of them can be two. That can be minus three. If both are minus three then the parentheses is zero. And then the whole product is zero."

Figure 8

Finding and Checking the Solutions to $(x-2)(x+3)=0$



This was also an example of the students applying creative mathematical reasoning to help them solve a novel (for them) problem by relying on elements of previous knowledge.

It was during this same task that they came to the realization on their own that it was much easier to solve a quadratic equation in factored form than in standard form, an insight which also seemed to indicate a transition towards the symbolic layer of generalized algebraic thinking.

272, S2: "This is harder. That one is much easier to see."

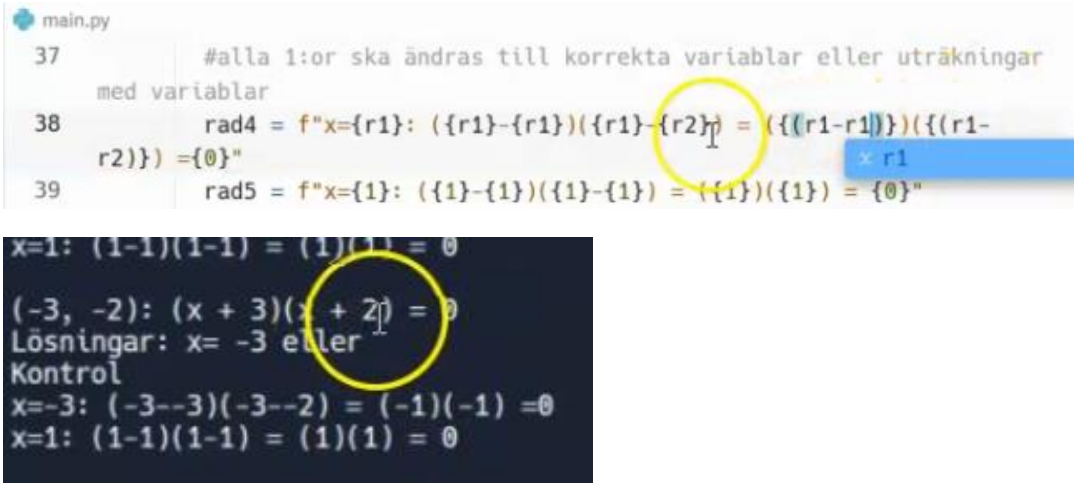
Their inexperience with programming was a problem from the start with task 4 on the computer. They often got lost in the details of programming which interfered with their ability to engage with the underlying mathematics (see section 13.5). However, it was still possible to identify small progress towards this construct motivated by

their attempts to understand how to write the program during this task (see Figure 9).

332, S1: “I'm thinking of exactly this, but instead of x we write r1. That's what I think.”
413, S2: “They come from there. That two should go in there. That two should go in there. And it's minus three which goes in here. Where the formula …”
459, S2: “And it's still r2 here.”
460, S1: “But it should be that since it's r2 there.”
461, S1: “That's r2, that's r1.”

Figure 9

Code and Output Showing Students Trying to Fix Their Code in Task 4



Note that the generalizations they were developing here to write the program required them to engage in creative mathematical thinking and to move towards the symbolic layer of generalized algebraic thinking.

In task 6 on paper they showed further construction of this knowledge element, once again by applying creative mathematical reasoning.

671, S2: “x is three … aha … x … x minus three … and x minus four.”

During task 6 on the computer they were able to show some insight connected to programming, though even here their inexperience was a major obstacle. They were trying to apply creative mathematical reasoning, but lack of knowledge around the details of programming was interfering with them understanding how to actualize their reasoning in this situation.

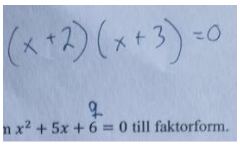
709, S1: "I think we should do it like this … it asks for an integer, that's the first one, so it's x minus plus the first integer times x plus minus the second integer."
749, S1: "It depends on if the number they say is positive or negative. Or yeah, it should be changed, so it has to be minus since that changes it."

Further construction was seen during task 7 on paper, though here there was no obvious connection to programming (see Figure 10).

802, S1: "Minus two, minus three."

Figure 10

Students Solve $x^2 + 5x + 6 = 0$ *by Factoring*

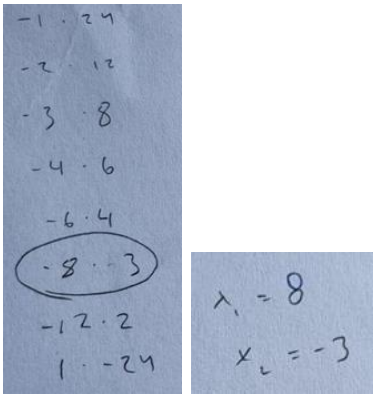


Similar construction could be seen in the final task on paper (see Figure 11). In both these examples, the students are also showing increased fluency with the symbolic layer of generalized algebraic thinking.

1136, S1: "We need to factor minus 24."
1143, S2: "That's it. Minus 8 times 3, that's minus 24. Minus 8 plus 3 is minus 5. That's the first one."
1151, S1: "x1 is 8 and x2 is 3."

Figure 11

Students Solve $x^2 - 5x - 24 = 0$ *by Factoring*



13.3.5 C_{st}: Connection Between Standard Form and Solutions

As all of the tasks were designed to try and help the students construct an understanding of the connection between the solutions to a quadratic equation and that equation in factored form, it is not surprising that epistemic actions connected to this construction

process can be seen throughout the lessons. The need to write a program to solve various tasks seems to have helped the students construct this knowledge element at different stages. At least to some degree.

During task 5 on the computer, the students seemed to be able to start using their understanding of factored form to start constructing this knowledge element.

517, S1: "Should we maybe have r1 here?"

In Figure 12 the students can be seen first testing 2 and -15 for an equation they worked out by hand, and then using that to start making the connection between the coefficients in standard form and the solutions to the equation.

Figure 12

Students Test 2 and -15

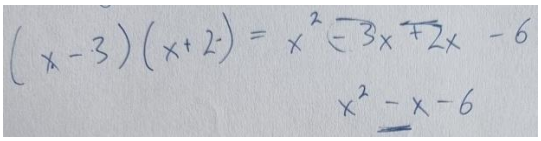
```
    s = get_standardform(2,-15)
    print("(x-3)(x+5)=", s)
    s = get_standardform(2,-15)
    print(s)

    rad4 = f"x={r1}: ({r1}-{r1})({r1}-{r2
{(r1-r1)*(r1-r2)}"
    rad5 = f"x={r2}: ({r2}-{r1})({r2}-{r2
r1)*(r2-r2)}"
```

Despite their programming difficulties, the need to program did seem to cause them to reflect on how to generalize the connection between factored form and standard form and in the process move towards the symbolic layer of generalized algebraic thinking facilitated by the use of creative mathematical reasoning. This involved them working with pen and paper to try and understand their own reasoning process (see Figure 13).

543, S2: "And then minus six. Then it's just minus one x."

Figure 13

Students Explore Their Process in Order to Write a Program



They struggled a lot with this task, but eventually started to make some progress and through that continue to construct this knowledge element through further application of creative mathematical reasoning, even though there are still some errors in their reasoning at this point. Here they also showed increased fluency with the symbolic layer of generalized algebraic thinking.

627, S1: "That was it. r1 times r2."
635, S2: "It's actually r1 plus r2."

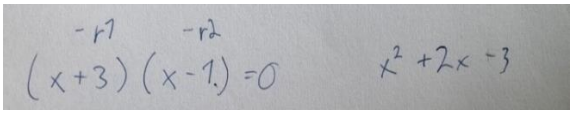
```
s=get_standardform(r1+r2, r1*r2)
```

They showed a better understanding of the connection between the coefficients in standard form and the solutions during task 6 on paper (see Figure 14).

690, S1 "So, what did we say? It was that times that."
691, S2 "Yeah."
692, S1 "It's that ..."
693, S2 "That plus that."

Figure 14

Students Connect Factored Form, Standard Form and the Solutions

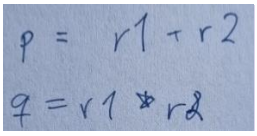


Further generalization was seen during task 7 on the computer (see Figure 15).

1070, S2 "Then ... ahh. Wait, wait, wait. p ... p is ... what was p? p was that. p is r1 plus r2 and q is r1 times r2."

Figure 15

Students Show a More Generalized Understanding of the Connection Between the Coefficients and the Solutions

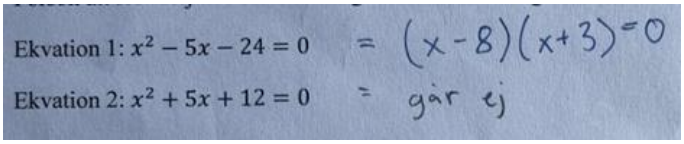


Here it seems that the need to write a program helped motivate them to move towards the symbolic layer of generalized algebraic thinking facilitated by creative mathematical reasoning, even if they are still struggling with the correct sign for the sum.

During the final task on paper, the students seemed to put it all together and solve one equation by factoring and determine that the second one is not factorable over the integers (see Figure 16).

Figure 16

Students Solve the Last Two Equations by Factoring Over the Integers



13.4 AiC Analysis of Intervention 3 Group 2

Table 6 below lists the relevant constructs, along with their connected epistemic actions, identified in this intervention. This includes constructs identified from the a priori analysis, as well as new constructs seen in the a posteriori analysis. The epistemic actions identified were recognizing (R), building-with (B), and constructing (C).

Table 6

Constructs Identified in Intervention 3 Group 2

Label	Description and comments
R_o B_o	The zero product property. Construction of this knowledge element was not identified during the lessons as the students seemed to already have a good understanding of the zero product property. However, they did recognize and build with this knowledge element while constructing other knowledge elements.
R_{fi} B_{fi} C_{fi}	This construct relates to the understanding of the factors of an integer, in particular as a procept. The students started with some basic constructs, but were able to develop them more during the tasks.
R_{mb} B_{mb} C_{mb}	How to multiply two binomials. This was a previous construct. However, it was not consolidated, so further construction was seen with it during the lesson, moving it towards consolidation.
R_{ffs} B_{ffs} C_{ffs}	The connection between an equation in factored form and the solutions to the equation. This was a new construct seen in the lessons.
R_{st} B_{st} C_{st}	This construct is an understanding of quadratic equations in standard form, in particular how the coefficients relate to the solutions and to factored form. This includes factoring it when possible. This was a new construct seen in the lessons.

Each of the following sections highlights relevant constructing actions connected to each specific construct. Evidence is also provided and discussed in cases where the act of programming appears to have facilitated the construction process. Example epistemic actions were chosen to exemplify the process for each particular construct. A more detailed coding of the epistemic actions for each construct can be found in Appendix 2 *Coded Transcription Lines*.

13.4.1 C_0: The Zero Product Property

The students showed some previous understanding that if the product of two numbers is zero then one of them must be zero. However, they were initially only certain about the *if* understanding, and not the *if and only if* understanding, indicating that this construct was not yet consolidated.

For example, from task 3 on the computer:

200, S2: "It's because as soon as it's zero times something, it's zero."

However, from task 4 on paper:

218, Me: "Is there another way to zero when multiplying two numbers?"
220, S1: "I don't know. We can test it ..."

Later during the same task, after further discussion, they seemed to construct a more solid *if and only if* understanding of this knowledge element.

234, S2: "No, the only way to do it is if one of the factors is zero."

13.4.2 C_{fi}: Factors of an Integer

The students had worked on the task connected to factoring an integer the week before the intervention and demonstrated some previously constructed knowledge here. One of the students had previous programming experience and had managed to complete a program able to factor positive integers before the intervention. However, all three students in the group struggled to extend their knowledge to a general method for factoring negative integers.

During task 2 on the computer they were able to recognize and build with previous constructs connected to factoring an integer, but I was not able to detect any construction of knowledge when it came to negative integers.

The need to program did seem to motivate them to begin to engage in creative mathematical reasoning, even if they did not get very far with it this time. For example:

112, S3: "One has to be negative, one has to be positive."

13.4.3 C_{mb}: Multiplying Two Binomials

Although the students had previous constructs connected to multiplying two binomials, they were seen to move towards consolidation during task 5 on paper. In task 3 on the computer they were able to recognize and build with constructs connected to understanding equations as part of the constructions process here. These early recognizing and building with actions seemed to be motivated by the need to program. For example, from task 3 on the computer:

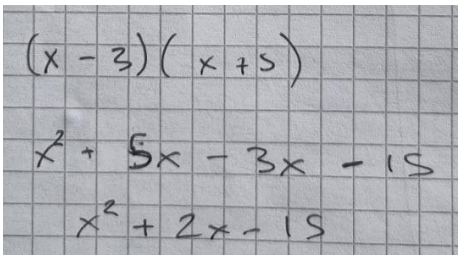
184, S2: "So, we can write that there inside parentheses …"
186, S1: "We can just take … we start with that."

The students successfully combined the linear terms in task 5 on paper (see Figure 17):

544, S2: "So it's basically x up to two plus 2x minus 15. There you have a simplification."
554, S1: "Okay, so we multiplied the numbers. And then we got x up to two plus 5x minus 3x minus 15. And then we simplified."

Figure 17

Multiplying $(x-3)(x+5)$ *and Successfully Combing the Linear Terms*



13.4.4 C_{ffs}: Connection Between Factored Form and Solutions

As all of the tasks were designed to try and help the students construct an understanding of the connection between the solutions to a quadratic equation and that equation in factored form, it is not surprising that epistemic actions connected to this construction process can be seen throughout the lessons. The need to write a program to solve various tasks seems to have helped the students construct this knowledge element at different stages. At least to some degree.

In task 2 on the computer they started to recognize and build with their knowledge of equations as part of their path towards constructing this knowledge element. Here it seemed that the need to program motivated them to engage in creative mathematical reasoning.

16, S1: "The number should be a variable."
47, S1: "I wrote in x here and then z divided by x."
68, S2: "I'm thinking that it might be a negative number. Since that's the negative number there."

Later, in task 3 on the computer, they started to recognize and build with the zero product property. The need to program here also seemed to motivate them to engage in creative mathematical reasoning.

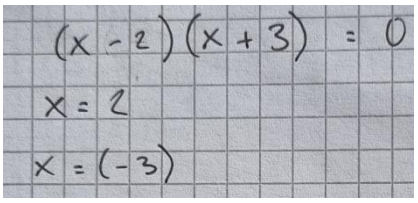
200, S2: "It's because as soon as it's zero times something, it's zero."

Initial construction actions could be identified in task 4 on paper (see Figure 18).

210, S3: "So there are two solutions."
211, S2: "There are two solutions."
212, S1: "Zero times everything is just zero."
213, S2: "x is equal to 2 or x is equal to minus 3."

Figure 18

Students Solve $(x-2)(x+3)=0$



Further construction of this knowledge element was seen in task 4 on the computer, where the need to program a general solution seemed to motivate the students to engage in creative mathematical reasoning which helped with the construction of new knowledge here.

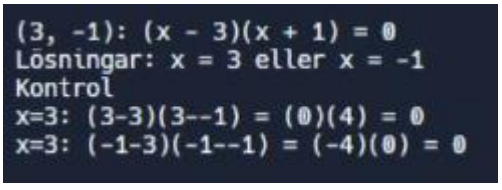
271, S2: "... the solutions are x is equal to minus 3 or x is equal to minus 1"
286, S1: "Because that's the opposite, right?"

They struggled with technical details of programming during this task, but also with a proceptual view of a quadratic equation in factored form as well as working completely within the symbolic layer of algebraic reasoning. However, as they continued to reason, they were able to make conceptual progress in both these areas and eventually complete the task (see Figure 19):

432, S2: "x is equal to ... r1 ..."
435, S1: "And that one should be ..."
436, S2: "Should be r2."
444, S2: "r1. And that one is r1. And that one is r2."
445, S1: "Yes. Then we want the same but we want to change ..."

Figure 19

Program Output Showing Progress on the Task

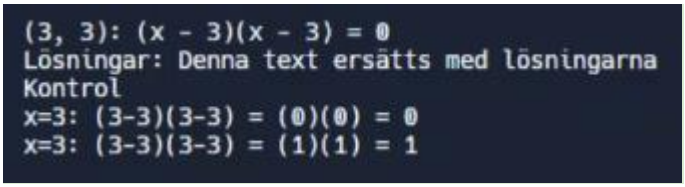


The programming aspects of the task were also seen to cause cognitive conflict in the students which then motivated further creative mathematical reasoning. This was seen to happen when they examined the output for "special cases", such as when both roots were the same. This initially looked like an error to them, and it took some effort and reasoning for them to work out that it was actually correct (see Figure 20).

483, S2: “What the fuck? What happened there? It’s 3 minus 3. How is that r1?”

Figure 20

Program Output After They Fixed First Line of Checking the Answer

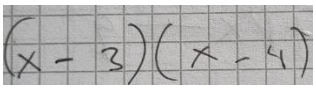


Additional construction of this knowledge element was seen in task 6 on paper where students showed increased fluency with the connection between the solutions to a quadratic equation and the equation in factored form. In this task, they show the ability to work within the contextual layer of algebraic thinking, though the task didn’t provide an opportunity to demonstrate fluency with the symbolic layer of algebraic thinking with this topic. They do, however, seem to show signs of understanding factored form as a procept in this task (see Figure 21).

718, S3: “So here we have the solutions for x.”
719, S2: “Then it’s just to put them in the values. It’s x minus 3 … x minus 4. Then we have factor form for it.”

Figure 21

The Students Write an Equation in Factored Form With Solutions $x = 3$ *and* $x = 4$



In task 6 on the computer, the need to program a general solution once again seemed to challenge the students’ understanding on the topic. They were seen to engage in creative mathematical reasoning in order to complete the program, as well as showing an increased fluency with the symbolic layer of algebraic reasoning.

793, S2: “So we should write … let’s use that number as an example. The program should basically print out x plus 2, x minus 1. It should then write x up to 2 and then … plus x and then minus 2.”
854, S2: “But that was standard form. If it’s factor form then it’s not r1 plus r2. it’s x plus r1 and x plus r2.” (*note she is confused here about the sigs a bit and how it connects to the function on the computer*)

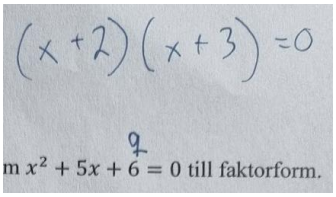
In the example above S2 is still showing some confusion about the signs and how exactly the solutions generally connect to factored form, even if the reasoning is mostly correct at this point.

In task 7 on paper, the students showed fluency in factoring a quadratic equation with integer solutions, basing the process on their understanding of the connection between factor form and coefficients in standard form (see Figure 22).

915, S1: "If they are multiplied we get that one, if they are added we get that one."
916, S2: "Then we've got it."

Figure 22

The Students Factor a Quadratic Equation



The second equation in task 7 on paper could not be factored over the integers. This initially caused cognitive conflict in the students which motivated them to engage in creative mathematical reasoning to understand what was going on. This process also allowed them to develop their ability to work within the symbolic layer of algebraic thinking and to increase their ability to view the equation as a procept and not just an indication of a procedure to follow.

927, S1: "No, it isn't minus. If it was minus it would be negative."
937, S1: "Both can't be negative because then it would be negative. If both were negative then that one would be negative and that one positive."
939, S3: "Is it even possible?"
940, S1: "I'm trying to think. I don't think it's possible."
941, S2: "It's not solvable."

The students showed similar construction actions during the final task on paper where they were once again given two equations in standard form, one with integer solutions and one without integer solutions. This time they were asked to solve them by factoring.

During this task they showed further construction of the knowledge element through the application of creative mathematical reasoning. However, as seen earlier, they still struggled with some of the details,

in particular the correct use of negative numbers and considering that, for example, 1 and 24 are also factors of 24. It is not possible to tell with certainty from this data if this was a context specific difficulty or if it was representative of general difficulties working with the symbolic layer of algebraic thinking.

1025, S2: "The factors are there … 24. It's either 4 times 6, 3 times 8. And minus 4, minus 6 and minus 3, minus 8. Their sum is 10. Their sum is 11. The sum of minus 4 plus minus 6 … is that the one we should use?"
1026, S3: "Minus 5."
1027, S2: "That one … minus 3 plus minus 8 is 5, or it's minus 5."

The excerpt above illustrates some of these errors, such as factoring 24 instead of -24, and then errors with adding negative numbers. However, it shows a general understanding of the concepts involved. They initially tried to factor $x^2 - 5x - 24$ as $(x - 8)(x - 3)$ but were soon able to determine that this did not work.

1034, S1: "I thought that could be positive and that could be negative."
1035, S2: "Does it matter?"
1040, S2: "Yeah, it does. It absolutely matters."

They now tried $(x + 8)(x - 3)$ which they figured out was still wrong. They were then able to factor it correctly and find the solutions from the factors (see Figure 23).

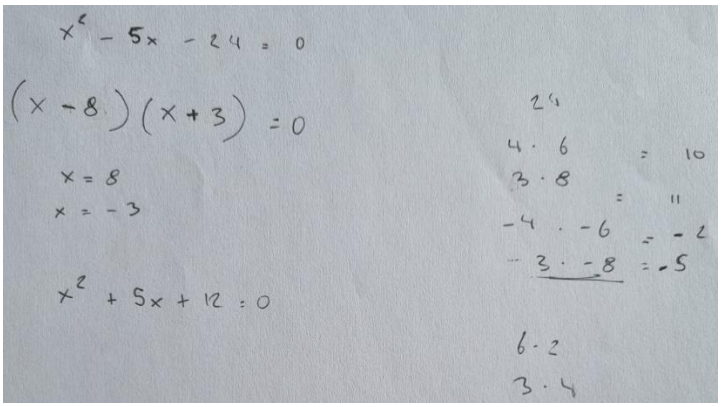
1045, S1: "So that should be minus and that should be plus."
1046, S2: "Exactly."
1050, S3: "Here you have x is 8, and here x minus 3."

This time they were quickly able to determine that the second equation could not be factored over the integers (see Figure 23).

1052, S2: "6 times 2 … there's 3 times 4. They all have to be positive because it is always positive. But none of these add up to 5. So it's not possible."
1053, S2: "Unless there's a negative."
1054, S2: "No, there isn't a negative because it's all positive. And no sum of factors is 5. So it's not possible.

Figure 23

Students Solve Quadratic Equations by Factoring Over the Integers



13.4.5 C_{st}: Connection Between Standard Form and Solutions

As all of the tasks were designed to try and help the students construct an understanding of the connection between the solutions to a quadratic equation and that equation in factored form, it is not surprising that epistemic actions connected to this construction process can be seen throughout the lessons. The need to write a program to solve various tasks seems to have helped the students construct this knowledge element at different stages. At least to some degree.

Since many of constructing actions for this knowledge element were the same as for the previous one, an attempt will be made to limit the amount of repetition here.

During task 5 on the computer, the students were able to recognize and build with their knowledge of how to multiply two binomials to start constructing this knowledge element. This construction process seemed to have been motivated by the programming requirements of the task.

Due to difficulties the students had with the technical details of programming, they were given skeleton code to modify instead of having to write the entire program from scratch (see Figure 24). This included a function that formatted an equation in standard form for them if they passed it the coefficients.

583, S1: “I wonder where 2x came from.”

Figure 24

Students Don’t Understand What the Functions Do

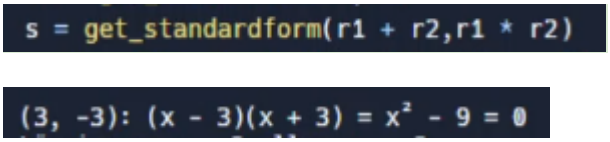
584, S2: “2x is … 5x minus 3x is 2x.”
597, S1: “So here is 2 and there is 2 and there is minus 15 and there is minus 15.”
598, S2: “Yes.”
599, S1: “So we took … we multiplied the. Wait, what did we do? To get 15 we can multiply r1 an r2.”
603, S1: “It’s q … q is r1 times r2. And p …”
608, S2: “r1 minus r2.”
609, S1: “Or add them. Because it’s 2. Because it’s … it’s negative. It’s addition.”

Once they understood how the function worked they began to engage in creative mathematical reasoning connecting the solutions to the coefficients. They didn’t have much trouble with the constant term, but struggled a bit with the sign difference for the coefficient of the linear term.

615, S2: “It works … for some. The one at the bottom.”

Figure 25

Students Struggle With the Linear Term

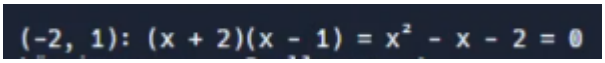


In the output example shown in Figure 25, their sign error is hidden which they interpreted as their idea working for some cases. They also thought it worked in some examples where it didn’t (see Figure 26):

624, S2: “But then it’s right.”

Figure 26

Students Don’t Notice the Sign is Wrong



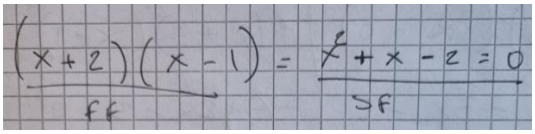
Since they weren’t sure, they checked the answer to this one with pen and paper (see Figure 27), engaging in more creative mathematical reasoning as well as showing fluency with the contextual layer of

algebraic thinking, even they were still struggling with the symbolic layer.

638, S1: "No, it should be plus there. It isn't."

Figure 27

Students Check the Programs by Hand

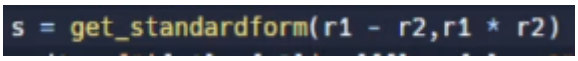


Even realizing their error they still had trouble understanding how to fix it at this point (see Figure 28).

660, S2: "Use subtraction."
661, S1: "Which one should be minus?"
662, S2: "Just use minus between them. We can see if it's correct."

Figure 28

Students Try and Fix the Sign Error

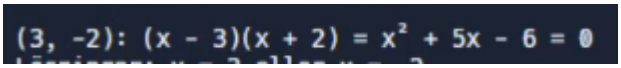


They realize they need to change the sign of the middle term, but don't understand how to do it (see Figure 29). Since the process to do this in the program is same as in mathematics in general, this seems to be a clear example of the need to program a general solution motivating their need to more fully construct this knowledge element though the process of creative mathematical reasoning. This is also increasing their fluency with the symbolic layer of algebraic thinking and helping them see an equation as a procept and not just a procedure to enact.

665, S2: "It's wrong. It's 5x here."

Figure 29

Students See That It Is Still Wrong



666, S1: "Yeah, I thought so since if it takes minus 2 minus 2 then it minus 3. So that doesn't work."
669, S1: "I wonder if it's so simple that we just … can you make them negative like that? You can. No, I need to use times."
671, S2: "What are you doing?"
672, S1: "I'm just changing from negative to positive. Or, it depends."

Figure 30

Students Find a Working Solution

```
s = get_standardform(-1 * r1 + r2 * -1,r1 * r2)
```

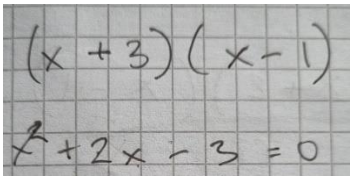
This final solution actually works, though is a bit awkward (see Figure 30). They understand that they need to change the sign of both variables, and also show and understanding that multiplying by -1 changes the sign and doesn't just make it negative.

During task 6 on paper they showed further construction of this knowledge element as well as showing a better understanding of standard form and factored form as procepts when they were able to convert from factored form to standard form without needing to multiply out the binomials (see Figure 31).

773, S2: "Plus 2x minus 3."
774, S1: "Plus 2x minus 3."

Figure 31

Students Easily Change From Factored Form to Standard Form



The need to program a general solution in task 6 on the computer seemed to motivate them to engage in creative mathematical reasoning which led to further construction of this knowledge element, as well as motivating them to increase their fluency with the symbolic layer of algebraic thinking.

878, S1: "So in standard we should just have p and q. Right?"
879, S2: "Yes. And that's how we get p and q. p is by multiplying."
894, S1: "Aha. You did it like that. You used what it says here and it works."

Figure 32

Students Correctly Complete the Program

```
r1 = int(input("Ange ett heltal: "))
r2 = int(input("Ange ett annat heltal: "))

standard = get_standardform(-(r1+r2),r1*r2)
faktor = get_faktorform(r1,r2)
```

In this task, they were able to correctly program the general relationship between the solutions and the coefficients of the equation in standard form. They also showed good symbolic fluency by using the minus sign to change the sign of the sum of r1 and r2, finally understanding that they did not need to subtract one from the other (see Figure 32). Importantly, they also showed an understanding that the minus sign did not make the expression negative but rather changed its sign.

Further construction of this knowledge element was also seen in task 7 on paper and the final task on paper. However, in these tasks the construction actions overlap with those already discussed in detail in the previous section and so will not be repeated here.

13.5 Summary of Results From Intervention 3

Results from intervention 3 provide evidence that programming tasks can help students learn to use the zero product property to solve quadratic equations. At a number of points throughout the tasks, the need to program appeared to motivate the students to engage in creative mathematical reasoning which helped them transition from a procedural understanding to a proceptual understanding of key concepts. However, their lack of programming experience and knowledge often seemed to get in the way of them making as much progress as was intended.

The beginning of this process could be seen when they tried to write a program to factor an arbitrary integer. When the students first started working on the factorization problem with pen and paper, they displayed factual algebraic thinking; they could perform the process with specific numbers, but had trouble describing it verbally. After they were given some help, they were able to start to describe the process verbally, though still in a contextually dependent manner. It was only when they had to implement the generalized procedure in a computer program that they seemed to start to take the final steps to partially freeing the symbolic representations from their context and thus exhibit symbolic algebraic thinking. Unfortunately, they were unable to complete the task due to lack of programming experience and so were never fully able to develop their reasoning here.

Although they were unable to write programs from scratch during the later tasks, they were able to complete partially written skeleton code. Even this partial programming process seemed to motivate the need to engage in creative mathematical reasoning which seemed to lead to the construction of the knowledge necessary to solve a quadratic equation by factoring and using the zero product property. Direct application of this construct was seen in the final task where they quickly and successfully solved a quadratic equation using this method with pen and paper, and identified a second equation as not having integer solutions.

Programming tasks that required the students to programmatically verify both solutions seemed to have been instrumental in helping them avoid the problem seen in literature where some students mistakenly check both solutions at the same time.

Programming tasks that required them to programmatically convert from factored form to standard form, and vice versa, and both finding solutions and creating equations from solutions, all seemed to contribute to the students understanding the structure of a quadratic equation, as well as using factoring and the zero product property. The most important part of these programs appeared to be discovering and encoding the exact relationship between the solutions to a quadratic equation and the coefficients of the equation in standard form (i.e., Vieta's formulas).

14 The Effects of Previous Programming Experience

This section explores how the students' previous programming experience, or lack thereof, influenced their ability to engage with the underlying mathematics in the tasks. The material presented in this section came to light during the detailed observation steps of the AiC methodology. Two broad categories of difficulties in engaging with the underlying mathematics were identified during an analysis of the students' comments and programming progress. The first category includes difficulties stemming from most of the students knowing too little programming. The second category includes difficulties stemming from some of the students having too much experience with programming.

The difficulties identified in the first category (described in section 14.1) were analyzed in relation to previous research into aspects of learning to program that novice programmers often struggle with (see section 6.2).

Note that AiC analysis is not used in this section, only its methodology. AiC analysis focuses on the construction of new knowledge while this section focuses on difficulties which might instead interfere with the construction of new knowledge.

14.1 Student Difficulties Stemming from Lack of Experience

Although one student in intervention 3 had some previous programming experience, the rest of the students had almost none. Because of this, both groups experienced difficulties with the technical details of programming which often interfered with their ability to engage with the underlying mathematics in the tasks.

Section 6.2 describes the most common difficulties novice programmers encounter according to the literature. Examples from the interventions have been chosen from three of the difficulties most frequently discussed in the literature.

The sections are presented in order of perceived impact on the students during the interventions, starting with the difficulties that appeared to have the greatest impact.

- Difficulties with variables and the notional machine (see sections 6.2.3 and 6.2.6). These two topics have been combined due to their extensive overlap and the difficulty in clearly delineating one from the other.
- Difficulties with loops (see section 6.2.7).
- Difficulties with syntax (see section 6.2.5).

14.1.1 Difficulties With Variables and the Notional Machine

As discussed in the literature (see section 6.2.3), misunderstanding the notional machine may underlie many of the other difficulties students have when learning to program. In addition, the correct use of variables is fundamental to programming as many other programming concepts build on the use of variables (see section 6.2.6). As it is not always possible to ascertain whether a particular difficulty is with understanding the notional machine or with understanding the correct use of variables, these issues will be discussed together in this section.

The students in intervention 3, group 1 showed trouble with variables and the notional machine right from the start. For example, at the beginning of one of their programs they typed:

```
int(tal**0.5)
```

Here they didn't seem to realize they needed to store the result in a variable for later use, a common problem seen in the literature among novice programmers.

This problematic understanding of how variables are used was also seen a short time later when they tried to use the following statement for their loop:

```
for tal in range(-tal, tal)
```

This is an example where the cognitive demands of programming prevented the students from adequately engaging with the underlying mathematics. They had previously come across the idea of only needing

to check up to the square root of an integer when testing for factors, but were unable to integrate that idea into the program without help.

They continued to have problems connected to ideas of the notional machine and how variables work even during the last lesson. For example, when trying to create a string that would print a quadratic equation in factored form with solutions $r1$ and $r2$, they first tried this:

```
s=(x-r1)(x-r2)
```

Here they seemed to expect the computer to understand what they really meant in the same way they might expect a teacher to interpret their notation.

The students in intervention 3, group 2 also had trouble with the notional machine. An early example of this was seen when they were working on the loop boundaries in a program to factor an arbitrary integer. They understood the modulo operator, but had included 0 in the range of factors to check and had trouble interpreting the error message they were getting.

136, S1: "Why is it not working here?"
140, S2: "Integer division by modulo zero."

They tried several times to fix the problem, but kept getting the same error and it took them a long time to find the problem. This could be seen as them assuming the computer "knew what they meant" when they specified the loop as a range up to the number being factored not realizing that in Python the `range` command starts at zero unless otherwise specified.

14.1.2 Difficulties With Loops

Difficulties understanding loops among novice programmers is also seen frequently in the literature (see section 6.2.7).

One example of this was with when group 2 in intervention 3 struggled with how to use the control variable for a loop in the loop itself. This could be seen when they were working on the program to factor an arbitrary integer.

In the example below, *h* is the control variable for the loop.

```
if tal%h==0:
    print(tal*h)
```

Not only were the students unable to provide a mathematical justification for multiplying a number by one of its factors, they did not even show any awareness that this is what they were doing. This indicates a misunderstanding of the role of the control variable *h* in this loop.

14.1.3 Difficulties With Syntax

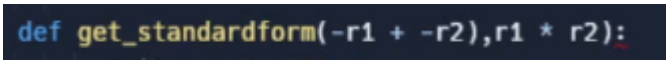
Although syntax errors also appear frequently in the literature (see section 6.2.5), they tend to be the easiest problems to fix and so are presented last in the list of example difficulties. The decision to include these difficulties was made because the students had enough trouble with syntax errors that it interfered with their ability to engage with the underlying mathematics at times. This included relatively simple syntax errors.

For example, the students in intervention 3, group 2 had trouble identifying a simple missing parenthesis (see Figure 33).

836, S1: "It's red everywhere."
842, S1: "I don't understand what the problem is."

Figure 33

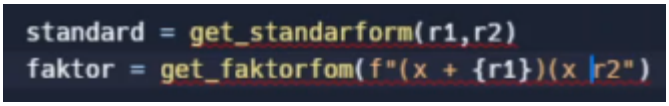
Students Don't See the Missing Parenthesis



These students also had trouble passing parameters to custom functions. For example, they confused the syntax for printing values of a variable with passing a variable to a function (see Figure 34).

Figure 34

Students Have Trouble With Function Calls



14.2 Student Difficulties Stemming from Previous Experience

The students in intervention 1 had extensive programming experience which made it easier for them to focus on completing the tasks instead of the technical details of programming. However, their fluency with programming was occasionally observed to interfere with their ability to engage with the underlying mathematics in the tasks. This occurred when the students were able to see a programming solution to tasks that would bypass the intended mathematics, even if the programming solution would take more time and more lines of code.

An example of this was seen when the students wrote a program to factor an arbitrary integer. They initially looped up to the integer itself, which created duplicates of all the integer pairs. For example, when factoring 6, the program would find both (2,3) and (3,2) as different pairs. There are several ways to fix this issue. A mathematical solution would be to loop up to the square root of the integer being factored. If the students had come to this realization, the only change necessary would have been adjusting the upper bound of the loop.

However, for these students, it was cognitively easier to try and remove all the duplicates from a list afterwards. Although this solution was programmatically complex, requiring a lot more time and effort and more lines of code, their programming experience made it easier for them to see and understand it compared to the solution which required more mathematical insight.

15 Discussions

The following sections discuss different aspects of this study.

- Section 15.1 discusses the methods used in this study.
- Section 15.2 discusses the results.
- Section 15.3 discusses ideas for future research.
- Section 15.4 is a broader, more general discussion of the material touched upon in this text.

15.1 Method Discussion

The use of the AiC methodology proved to be a powerful tool in this study. Although AiC does not seem to be a widely used methodology, it has appeared in research looking at how students learn mathematics (Kouropatov & Dreyfus, 2017; Nurhasanah et al., 2017; Nurhasanah & Sumekar, 2019; Ozmantar, 2004; Stehlikova, 2003; Tabach et al., 2017) including specifically looking at the use of programming to teach mathematics (Anabousy et al., 2023).

AiC involves identifying and coding all the small details that comprise the process of constructing new mathematical knowledge. This type of analysis allowed me to gain a rich understanding of the students' reasoning processes. I was not only able to see where they made progress and where they ran into obstacles, but I was also able to identify many of the underlying reasons. Furthermore, the comprehensive transcriptions and coding details provided at the end of this text increase the transparency of the whole process. All of which contributes to increasing the validity and reliability of this study.

The use of AiC also highlighted subtle complications in how knowledge is constructed. In particular, that the construction of knowledge does not seem to be a linear process, but more resembles a web of connections. Instead of just creating new knowledge constructs sequentially, students would at times need to first reinforce previous constructs (i.e., move them towards consolidation) before then continuing on to construct the new knowledge element. This was seen, for example, in the first intervention when the students showed further construction of knowledge connected to equations (C_e) and then used

that new knowledge construct within the same lesson sequence to help construct other knowledge elements.

The methodology proved useful in other ways as well, independent of the coding and analysis. For example, the detailed data gathering allowed me to identify where too little or even too much programming experience prevented students from engaging with the underlying mathematics (see section 14).

Task design was another essential aspect of this study's methodology as it used a task-based interview method (Maher & Sigley, 2020). In addition to using research-based design principles for the tasks (see section 9.3), the tasks were also discussed in research collaboration groups where I received helpful advice and suggestions at a number of different points in the study.

15.1.1 Knowledge Construct Granularity

An inherent complication with AiC arises when deciding on the granularity of the knowledge constructs of interest. For example, this study included a knowledge construct pertaining to general knowledge about equations and expressions (C_e) as well as one pertaining to specifically multiplying two binomials (C_{mb}). However, the second construct here could be seen as a subset of the first one.

In this particular case, the data from the first intervention seemed to support this division of constructs. While the students were constructing a more proceptual understanding of equations as a whole, they still struggled with a proceptual understanding of the product of two binomials. This is another indication of the complexity and non-linearity of these knowledge constructs. It is also an indication of how a non-consolidated construct, in this case the one connected to equations and expressions, can be disrupted by situations that seem to encourage the students to lean more heavily on a purely procedural understanding

As a general principle, it seems useful to be as fine-grained as possible in looking at constructs, at least in order to increase the chances of identifying important epistemic actions. However, analyzing too many small constructs could quickly get out of hand and have the opposite effect, perhaps causing the researcher to "miss the forest for the trees".

There are no clear guidelines for sorting out this dilemma in any of the relevant literature, and so perhaps it is a matter of experience and something that can be best worked out in the a posteriori analysis as was done with the example above.

15.1.2 Methodology Problems

The methods used in this study were not problem-free. AiC is an incredibly time consuming methodology. All potential gains in detail and thoroughness have to be balanced against the cost in time and effort. In order to fully use this methodology, a lot of data needs to be generated and analyzed. A possible fix for future researchers could be to have a much narrower focus in the tasks being looked at. Fewer and simpler tasks may reduce the data collected enough to better balance the cost in time and effort.

Another complication with using AiC was connected to who should run the lessons. According the authors of AiC (Dreyfus & Kidron, 2014), the preferred method is to have the students' ordinary mathematics teacher run the lessons in a normal classroom setting. This turned out to be difficult to achieve as only a few students wanted to participate and the teachers had limited time to help. This forced me to take on the role as both researcher and teacher.

15.1.3 Practical and Ethical Problems

Although not specifically connected to AiC, I did encounter a number of practical problems as well as some potential ethical issues which are worth discussing.

Right from the start I ran into problems finding students and teachers willing to participate in the study. In the end, the only students who were motivated enough to volunteer were all academically motivated and were on track to receive top grades in mathematics. If this study had found a strong positive result, this fact could have been considered a threat to the reliability of the results.

Lack of time during the interventions themselves was also a problem. The time issue was not as apparent during the first intervention due to the students' previous experience with programming. This contributed to me underestimating how much extra time the inexperienced

students in the third intervention needed to complete the tasks. Although fewer tasks may have mediated this problem, that had to be weighed against choosing tasks necessary for constructing the required knowledge elements. This raises the question of whether the problem could be fixed with a different choice of tasks or if the problem actually lies in the whole idea of trying to learn this type of mathematics through programming.

Another practical problem was connected to obtaining good data to analyze. Data gathering consisted of two parts, screen recordings and audio recordings, with a different set of concerns for each one. When the students were able to use a computer I provided with software I had previously installed and checked, the screen recording worked as expected. This included additional audio recording through the computer which could be used to help transcribe unclear sections of the purely audio recordings. However, on several occasions, some of the students had to use their own computer since I only had one computer available to give to the students. Not only did this mean trying to find effective free software for them to download and install, but it also meant using valuable time during the intervention to get it working. In the end, the free software didn't work very well, and none of the screen recordings from students' own computers were of good quality. Unfortunately, this was not apparent until after the interventions were finished.

A bigger problem was discovered with the audio recordings from intervention 2. This was the first intervention with two separate groups, and I had seated them near each other so that they could collaborate as needed. I was unaware how sensitive the audio recording devices were, and in the end it was not possible to adequately determine which student was speaking at any given time. This, in effect, rendered all the data from intervention 2 unusable for detailed analysis.

A potential ethical issue also arose during intervention 2. It was apparent that most of the participating students were not very interested in being there. They lost focus often, and would sometimes wander away from their computers for a while. They were supposed to be there completely voluntarily and so I had no authority to encourage

them to work on the tasks. They did stay in the room since it was during their normal mathematics lesson and their teacher had made clear that they had to be with me the whole time to be marked as “present” for the lesson. I also had the impression that the teacher pushed them to participate and the degree to which they truly volunteered was unclear. Being new to field research, I was unsure how best to deal with such a situation and the ethical questions that arose, so perhaps it is for the best that the data was not useable in the end.

15.2 Results Discussion

The aim of this study was to explore whether or not the act of programming itself, as opposed to the results or output of a program, can help students develop a deeper understanding of mathematics. To this end, an attempt was made to answer the following two research questions:

RQ 1. In what ways can programming tasks facilitate student learning and understanding of how to use the zero product property to solve quadratic equations?

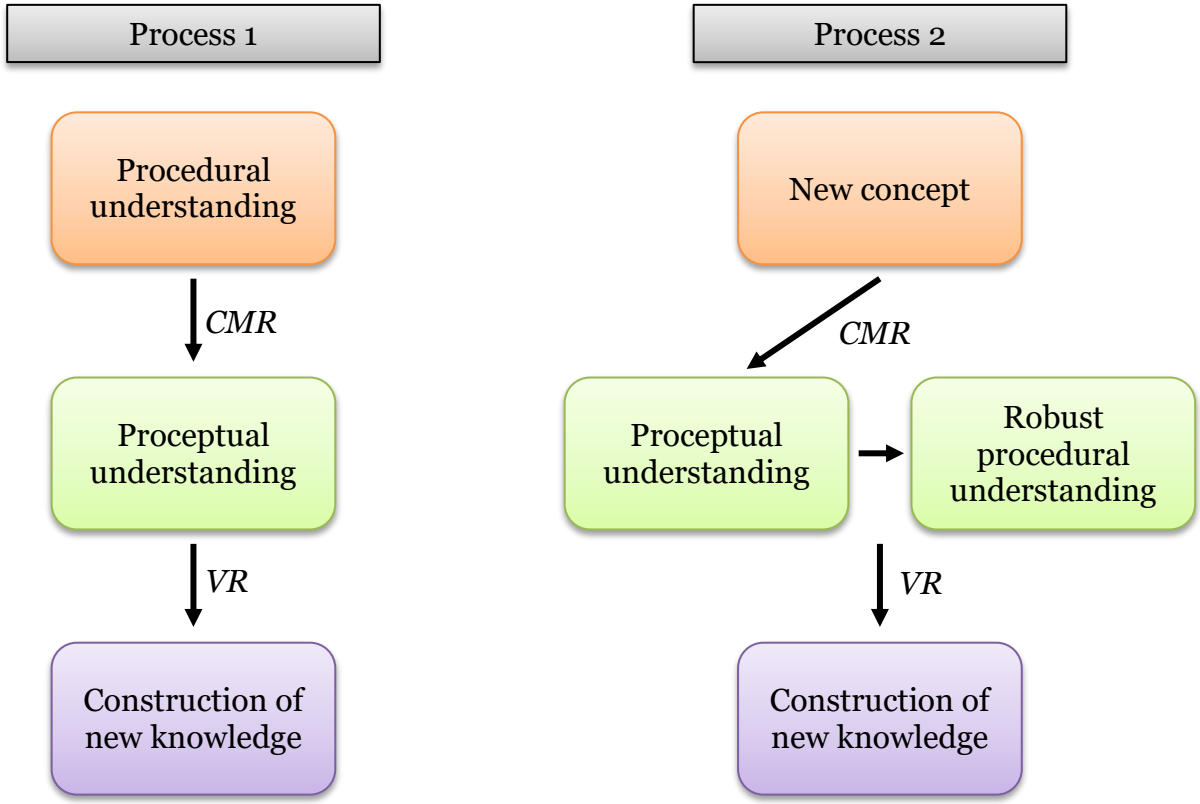
RQ 2. Which aspects of the programming tasks are helpful in facilitating the avoidance of common misunderstandings previously identified in the literature which are associated with using the zero product property to solve quadratic equations.

15.2.1 Findings in Support of the Aim

Two main processes were identified that support the aim of this study. Both processes involved using Lithner’s (2008) version of creative mathematical reasoning (CMR), motivated by the need to complete a programming task, to develop a proceptual understanding of the underlying mathematics which led to the construction of new knowledge through the process of vertical reorganization (VR) as described by Dreyfus and Kidron (2014). These processes are illustrated in Figure 35.

Figure 35

The Two Main Construction Processes Seen in the Findings



The first process was observed when the students started with some degree of previously learned procedural knowledge connected to the task, but soon realized that this was not enough to write the program. This proved to be the case even when they demonstrated a high degree of fluency with the procedure. For example, none of the students had any trouble factoring any positive integer they were given. However, despite this fluency with the procedure, they were initially unable to explicitly explain what they were doing, which prevented them from being able to write a program that could factor an arbitrary integer.

Once the students were confronted with this impasse, they began to reflect on their understanding of the mathematics in discussions with each other. This process of reflection and discussion matches Lithner's (2008) description of creative mathematical reasoning and seems to have led the students to move away from a purely procedural understanding of the mathematics towards a more proceptual understanding (Gray & Tall, 1994).

The second process was observed when the students had no previous knowledge of the mathematical concept being explored (e.g., factoring quadratic equations). As with the first process, the need to complete programming tasks prompted the students to begin to reason with each other about the underlying mathematics. The difference here was that the students seemed to develop a proceptual understanding of the mathematics before a purely procedural one. In addition, the procedural knowledge they did develop appeared to be more be more robust as the students were able to recover forgotten procedural details based on their understanding of the mathematics.

15.2.2 Findings in Support of RQ 1

As discussed in the previous section, the findings support the idea that the need to complete programming tasks can motivate students to develop a more proceptual understanding of the underlying mathematics through engaging in creative mathematical reason. This was seen specifically in the interventions with regards to using the zero product property to solve quadratic equations.

Difficulty using the zero product property appears frequently in the literature, usually alongside problems with factoring, as a common obstacle to solving quadratic equations (e.g., Didiş et al., 2011; Didiş & Erbas, 2015; López et al., 2016; Memnun et al., 2015; Vaiyavutjamai & Clements, 2006).

All of the students demonstrated an understanding that if you multiply a number by 0 then the result must be 0. I will refer to this as the *if* understanding of this property. However, all the students initially showed difficulty understanding that if the product of two factors is 0, then at least one of the factors must be 0. I will refer to this as the *if and only if* understanding.

An example of how this difference in understanding may manifest itself can be seen in the literature where students were asked to solve a factored quadratic equation such as $(x - 2)(x - 3) = 0$. Some students first multiplied out the factors and then either used the quadratic formula or tried to refactor it. This seems to indicate that these students did not fully understand that if the product of two numbers is 0, then

one of them must be 0 (e.g., Hu et al., 2021; Makonye & Nhlanhla, 2014; O'Connor & Norton, 2016).

This particular problem was not seen with the students in the interventions, which could be attributed to the fact that the students had already developed an *if and only if* understanding of the zero product property by the time they were asked to solve quadratic equations. Some of the students even made specific spontaneous comments about how much easier it was to solve equations in factored form.

In order to use the zero product property to solve a quadratic equation, the equation first has to be factored. This process depends on understanding the connection between the solutions to the equation and the equation in factored form as well as the equation in standard form. Completing the programming tasks appeared to help the students develop an understanding of these connections, even though they initially struggled with the connection between the solutions and the linear term in standard form.

After they had completed all the tasks, the students had no trouble using pen and paper to determine if a quadratic equation had integer solutions, and if it did, they were able to factor and solve it using the zero product property.

15.2.3 Findings in Support of RQ 2

My second research question dealt with how to design tasks specifically for learning to solve quadratic equations using the zero product property. The task elements that appeared to be most important in the interventions were those forcing the students to make their understanding explicit, in particular with regards to developing an *if and only if* understanding of the zero product property as well as a proceptual understanding of Vieta's formulas.

The importance of the students deriving Vieta's formulas for themselves and then making the knowledge explicit could be seen most clearly in intervention 3. Due to the students' lack of programming experience in this intervention, they were provided with an almost complete program for the task where they were expected to construct this knowledge. The students only had to modify a small part of the

program which involved them programmatically connecting the solutions to a quadratic equation to the coefficients of the equation in standard form. However, even this small amount of programming seemed to be enough to drive the construction of important new knowledge.

When completing later tasks, the students relied heavily on this knowledge construct, often needing to first reinforce their understanding of it before then using it to construct new knowledge elements. The fact of their frequent return to reasoning about this construct before using it helps to show not only its importance but also its difficulty.

Based on these findings, it may be possible to make an educated guess as to the more general answer to the question of how to design programming tasks to help students learn any mathematics. The focus would seem to need to be on critical aspects of the desired mathematical knowledge constructs, but in such a way that the students would need to explicitly describe their reasoning process before being able to write the program. This could help students first develop a proceptual understanding of the mathematics before a purely procedural one, as seen in this study. This would be in contrast to first learning a procedure and then later trying to understand it.

Although all of this lines up with the task design principles used in this study (see section 9.3), the actual implementation is unlikely to be obvious in most cases and would probably require some trial and error to make it work.

15.2.4 Findings not in Support of the Aim, RQ 1 or RQ 2

While the need to complete the programming tasks did seem to motivate these students to start the process of engaging in creative mathematical reasoning, it is not clear from these findings if this is a generally useful approach.

To start with, all of the participants were top students in mathematics. This means that, based on this study, it is not possible to make any claims as to whether or not similar results would have been seen with students who struggle with mathematics. For example, it could be the case that students in this study had a particularly low threshold for

engaging in creative mathematical reasoning which would imply that the same effect might not be seen in students who have a harder time in general with creative mathematical reasoning.

The findings also indicate that even if it might be possible for programming tasks to help students learn mathematics, this may not currently be a functional method in Swedish schools. In fact, an argument can even be made that it is unlikely to be a functional method for many years to come.

Several other researchers have previously warned about the risks of learning mathematics through programming. If the students consider the programming tasks to be too hard, they may lose interest in the mathematics being studied Dohn (2020). They also risk getting lost in the technical details of programming and losing focus on the mathematics (Borg, 2021; Ke, 2014). Both of these issues were seen in intervention 3.

This is not surprising since many researchers consider learning programming to be cognitively difficult (e.g, Berssanette & De Francisco, 2022). In fact, finding ways to specifically reduce the cognitive load when learning to program has been the focus of a number of recent studies (Berssanette & De Francisco, 2022; Choi & Choi, 2024). The split-attention principle may make matters even worse for students trying to learn both mathematics and programming at the same time. For example, Ayres and Sweller (2021) found that dividing a student's attention between different details in a problem, even by seemingly small degrees, could negatively influence their performance. This difference could be as simple as writing the angles inside a figure instead of under it. By comparison, splitting attention between new mathematical concepts and new programming concepts would seem likely to incur a much larger additional cognitive load.

15.3 Future Research

If mathematics teachers (at least in Sweden) will continue to be required to teach programming alongside mathematics, new ideas and methods need to be found to effectively integrate these two subjects. If this doesn't happen, there seems an obvious risk that programming will just be a separate extra subject taught in mathematics lessons. This is

clearly problematic as the mathematics content has not been reduced, nor has more time been added. Under these conditions it would be difficult to see programming as anything other than an obstacle to learning mathematics.

Even if my study failed to identify a novel way to integrate mathematics and programming with novice programmers, there may be reasons to conduct more research in this area. Perhaps another researcher will come up with better task designs that lead to the desired outcome. Perhaps clever integration of AI will be the solution.

This study's findings suggest another potential line of interesting research. The main way programming seems to help students learn mathematics is by motivating them to make their reasoning and procedures explicit to themselves and to learn to generalize them. This idea is not dependent on programming, and perhaps new research into ways to encourage this with pen and paper may produce better results.

15.4 Final Thoughts

It seems hard to argue with the value of computers and programming in mathematics *in general*. Teachers, researchers, and many others frequently use and appreciate a computer's ability to create visualizations of difficult mathematical ideas. Computers can run impressive simulations, they can rapidly perform complex calculations and they are even being used to prove mathematical theorems (e.g., Pantsar, 2024). The sophistication of computer programs used in education, research and engineering continues to improve.

However, writing a program is not the same thing as using one. Nor is any knowledge of programming necessary to use the vast majority of ready-made computer programs available. This is analogous to the fact that I don't need to know how my car works "under the hood" to be able to drive it effectively. In fact, as modern cars get more and more sophisticated, I seem to need to know less and less about how they work. In the same way, as user interfaces get "friendlier", the user needs to know less and less about the programming behind them.

While it may be true that programming is a useful and commonplace skill for those seeking advanced mathematics or science degrees, this can't be said to apply to the majority of students in primary and

secondary school. Most of these students will remain novice learners at best, only learning to program as part of a course requirement, such as within mathematics courses in Sweden today.

When considering how realistic it is to imagine a society where the vast majority of students are competent programmers, it may be worth remembering that humans have been teaching mathematics to students for thousands of years, and yet, still today, most students struggle with mathematics. Teaching programming is a much more recent human endeavor, with its own complex set of didactical challenges. To expect a general level of competence with programming in the near future, after failing to achieve the same with mathematics for thousands of years, seems unrealistic.

It also seems worthwhile to consider the experiences of mathematics teachers in Sweden in this context. After the changes were made to the curriculum, mathematics teachers, without any formal training, were suddenly expected to teach programming as part of their mathematics lessons, something they were not prepared for (Vinnervik, 2022). Looking at the views of mathematics teachers, Kilhamn, Bråting, et al. (2021) found that some teachers, especially those who already had some programming experience, were not entirely negative to the idea of students learning programming, although it was not clear to them how they could effectively combine it with mathematics. In the end, the authors conclude that "if the main purpose is to learn to use a new tool, it is questionable if the time spent on programming is wasted from a mathematics education point of view" (p.175). In other research, the same authors found that about a third of the mathematics teachers they looked at taught programming with no connection to mathematics, and about another third included mathematics as context, but without any ambition for the students to actually learn more mathematics through the programming activities (Kilhamn, Ronaldsson, et al., 2021). The latter case is akin to the common problem of introducing inauthentic contexts in mathematics word problems in order to try and connect mathematics to reality (e.g., Boaler, 1993).

To sum up, students are struggling to learn programming and mathematics teachers are struggling to integrate it into their mathematics teaching.

Looking to the future, it seems that the current explosion in easy access to AI software may dramatically change the programming landscape.

Currently, many of the tasks given to the students in the interventions can be solved by typing in natural language questions into one of the available online AI tools. The following is an example using ChatGPT 4.0.

Instruction given to the AI: *Create a Python function that takes an integer as a parameter and returns a list of all factor pairs of that integer, both positive and negative, excluding duplicates.*

Here is the program provided by ChatGPT:

```
def find_unique_factor_pairs(n):
    factor_pairs = []

    # Search for factors up to the absolute square root of n.
    # This ensures that we only add each pair once.
    for i in range(1, int(abs(n)**0.5) + 1):
        if n % i == 0:
            # If i is a factor, then n//i is also a factor.
            # Include both positive and negative factor pairs.
            factor_pairs.append((i, n//i))
            factor_pairs.append((-i, -n//i))

    return factor_pairs

# Testing the function
n = int(input("Enter an integer: "))
print(f"Unique factor pairs of {n} are:", find_unique_factor_pairs(n))
```

Besides including comments in the program, the AI also provided a short explanation: *This function finds factor pairs by checking divisors only up to the square root of the number. This minimizes redundancy, ensuring that each pair is added only once.*

While it may seem at first glance that AI is a shortcut that the students can use to bypass their own reasoning, it may also turn out to be a useful tool that can be integrated into programming lessons. Considering that the major obstacle encountered in the interventions described in this text was the difficulty students had learning to

program, it is not hard to imagine that a tool like ChatGPT could be used in some way to ameliorate this difficulty.

As a final note, I leave the reader with a short excerpt from intervention 3, group2:

106, S2: “A lot of programming is not mathematically correct.”
107, S1: “No, I know.”

References

Adler, J., & Davis, Z. (2006). Opening Another Black Box: Researching Mathematics for Teaching in Mathematics Teacher Education. *Journal for Research in Mathematics Education, 36*(4), 270-296.

Akpinar, Y., & Aslan, Ü. (2015). Supporting Children's Learning of Probability through Video Game Programming. *Journal of Educational Computing Research, 53*(2), 228-259. https://doi.org/10.1177/0735633115598492

Allaire, P. R., & Bradley, R. E. (2001). Geometric Approaches to Quadratic Equations from Other Times and Places. *The Mathematics Teacher, 94*(4), 308-319. https://doi.org/10.5951/MT.94.4.0308

ALLEA. (2017). *The European Code of Conduct for Research Integrity* (Revised ed.). ALLEA - All European Academics. https://allea.org/code-of-conduct/

Anabousy, A., Daher, W., & Bassan-Cincinatus, R. (2023). Scratch as an Environment for Learning the Coordinate System by Elementary School Students. *Education Sciences, 13*(7), 724. https://doi.org/10.3390/educsci13070724

Ayres, P., & Sweller, J. (2021). The Split-Attention Principle in Multimedia Learning. In R. E. Mayer & L. Fiorella (Eds.), *The Cambridge Handbook of Multimedia Learning* (pp. 199–211). Cambridge University Press.

Babori, A. (2022). A Didactic Study of an Algorithmic and Programming Mooc: Learning Strategies Adopted by Students and Their Difficulties. *International Journal of Emerging Technologies in Learning (iJET), 17*(19), 259-277. https://doi.org/10.3991/ijet.v17i19.32161

Baga, E. (2017). Arithmetical Algebra in the Islamic History of Mathematics and Its Peak in the 9th/15th Century: Ibn Al-Hāʾim's Al-Mumtiʿ. *Nazariyat İslam Felsefe ve Bilim Tarihi Araştırmaları Dergisi (Journal for the History of Islamic Philosophy and Sciences), 3*(2), 69-123. https://doi.org/10.12658/Nazariyat.3.2.M0009en

Bardini, C., Radford, L., & Sabena, C. (2005). Struggling with Variables, Parameters, and Indeterminate Objects or How to Go Insane in Mathematics. In H. L. Chick & J. L. Vincent (Eds.), *Proceedings of the 29th Conference of the International Group for the Psychology of Mathematics Education* (Vol. 2, pp. 129-136). University of Melbourne. https://www.emis.de/proceedings/PME29/PME29CompleteProc/PME29Vol2Adl_Fre.pdf

Barry, P. (2008). Factorizing Quadratics Again! *Mathematics in School, 37*(1), 26-27. https://www.jstor.org/stable/30216094

Bayman, P., & Mayer, R. E. (1983). A Diagnosis of Beginning Programmers' Misconceptions of Basic Programming Statements. *Communications of the ACM, 26*(9), 677-679. https://doi.org/https://doi.org/10.1145/358172.358408

Bennedsen, J., & Caspersen, M. E. (2007). Failure Rates in Introductory Programming. *The SIGCSE Bulletin, 39*(2), 32-36. https://doi.org/10.1145/1272848.1272879

Bennedsen, J., & Caspersen, M. E. (2019). Failure Rate in Introductory Programming: 12 Years Later. *ACM Inroads, 10*(2), 30-36. https://doi.org/10.1145/3324888

Benton, L., Hoyles, C., Kalas, I., & Noss, R. (2016). Building Mathematical Knowledge with Programming: Insights from the Scratchmaths Project. In A. Sipitakiat & N. Tutiyaphuengprasert (Eds.), *Constructionism in Action 2016: Conference Proceedings* (pp. 26-33). Suksapattana Foundation.

Benton, L., Hoyles, C., Kalas, I., & Noss, R. (2017). Bridging Primary Programming and Mathematics: Some Findings of Design Research in England. *Digital Experiences in Mathematics Education, 3*(2), 115-138. https://doi.org/10.1007/s40751-017-0028-x

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

Blikstein, P. (2018). *Pre-College Computer Science Education: A Survey of the Field*. Google LLC. https://goo.gl/gmS1Vm

Boaler, J. (1993). The Role of Contexts in the Mathematics Classroom: Do They Make Mathematics More "Real"? *For the Learning of Mathematics, 13*(2), 12-17.

Borg, A. (2021). *Designing for the Incorporation of Programming in Mathematical Education : Programming as an Instrument for Mathematical Problem Solving* [Licentiate dissertation, Karlstad University]. http://kau.diva-portal.org/smash/record.jsf?pid=diva2%3A1584627&dswid=-3447

Borg, A., Fahlgren, M., & Ruthven, K. (2020). Programming as a Mathematical Instrument: The Implementation of an Analytic Framework. In A. Donevska-Todorova, E. Faggiano, J. Trgalova, Z. Lavicza, R. Weinhandl, A. Clark-Wilson, & H.-G. Weigand (Eds.), *Mathematics Education in the Digital Age (Meda) Proceedings* (pp. 435-442).

Bossé, M. J., & Nandakumar, N. R. (2005). The Factorability of Quadratics: Motivation for More Techniques. *Teaching Mathematics and its Applications, 24*(4), 143-153. https://doi.org/10.1093/teamat/hrh018

Bosse, Y., & Gerosa, M. A. (2017). Why Is Programming So Difficult to Learn? Patterns of Difficulties Related to Programming Learning Mid-Stage. *ACM SIGSOFT Software Engineering Notes, 41*(6), 1-6. https://doi.org/10.1145/3011286.3011301

Brennan, K., & Resnick, M. (2012). *New Frameworks for Studying and Assessing the Development of Computational Thinking.* Proceedings of the 2012 Annual Meeting of the American Educational Research Association, Vancouver, Canada.

Bråting, K., & Kilhamn, C. (2020). Exploring the Intersection of Algebraic and Computational Thinking. *Mathematical Thinking and Learning*, 1-16. https://doi.org/10.1080/10986065.2020.1779012

Bråting, K., & Kilhamn, C. (2022). The Integration of Programming in Swedish School Mathematics: Investigating Elementary Mathematics Textbooks. *Scandinavian Journal of Educational Research, 66*(4), 594-609. https://doi.org/10.1080/00313831.2021.1897879

Bråting, K., Kilhamn, C., & Rolandsson, L. (2020). Integrating Programming in Swedish School Mathematics: Description of a Research Project. In *Sustainable Mathematics Education in a Digitalized World: Proceedings of Madif 12* (pp. 1-10). Uppsala University. http://uu.diva-portal.org/smash/record.jsf?pid=diva2%3A1432043&dswid=-451

Buteau, C., Muller, E., Gueudet, G., Mgombelo, J., & Sacristán, A. I. (2019). University Students Turning Computer Programming into an Instrument for "Authentic" Mathematical Work. *International Journal of Mathematical Education in Science and Technology*, *51*(7), 1020-1041. https://doi.org/10.1080/0020739X.2019.1648892

Byckling, P., & Sajaniemi, J. (2006). Roles of Variables and Programming Skills Improvement. *ACM SIGCSE Bulletin*, *38*(1), 413-417. https://doi.org/10.1145/1124706.1121470

Calao, L. A., Moreno-Léon, J., Correa, H. E., & Robles, G. (2015). Developing Mathematical Thinking with Scratch: An Experiment with 6th Grade Students. In G. Conole, T. Klobucar, C. Rensing, J. Konert, & É. Lavoué (Eds.), *Design for Teaching and Learning in a Networked World - 10th European Conference on Technology Enhanced Learning, Ec-Tel 2015, Toledo, Spain, September 15-18, 2015, Proceedings* (pp. 17-27). Springer.

Cetin, I., Sendurur, E., & Sendurur, P. (2014). Assessing the Impact of Meta-Cognitive Training on Students' Understanding of Introductory Programming Concepts. *Journal of Educational Computing Research*, *50*(4), 507-524. https://doi.org/10.2190/EC.50.4.d

Chemla, K. (1993). Algebraic Equations East and West until the Middle Ages. In K. Hashimoto, C. Jami, & L. Skar (Eds.), *East Asian Science: Tradition and Beyond: Papers from the Seventh International Conference on the History of Science in East Asia* (pp. 83-89). https://halshs.archives-ouvertes.fr/halshs-00865002

Chiodini, L., Santos, I. M., Gallidabino, A., Tafliovich, A., Santos, A. L., & Hauswirth, M. (2021). A Curated Inventory of Programming Language Misconceptions. In *Proceedings of the 26th Acm Conference on Innovation and Technology in Computer Science Education V. 1, Acm* (pp. 380-386). https://doi.org/10.1145/3430665.3456343

Choi, W. C., & Choi, I. C. (2024). Investigating the Effect of the Serious Game Codecombat on Cognitive Load in Python Programming Education. In *2024 IEEE World Engineering Education Conference (Edunine)* (pp. 1-6). https://doi.org/10.1109/EDUNINE60625.2024.10500551

Clark-Wilson, A., Noss, R., Hoyles, C., & Benton, L. (2019). Key Factors for Successfully Embedding a Programming Approach to the Primary Maths Curriculum at Scale. In U. T. Jankvist, M. v. d. Heuvel-Panhuizen, & M. Veldhuis (Eds.), *Eleventh Congress of the European Society for Research in Mathematics Education* (pp. 2681-2688). Freudenthal Group. https://hal.archives-ouvertes.fr/hal-02417059

Clements, D. H. (2000). From Exercises and Tasks to Problems and Projects: Unique Contributions of Computers to Innovative Mathematics Education. *Journal of Mathematical Behavior*, *19*(1), 9-47. https://doi.org/10.1016/s0732-3123(00)00036-5

Cline, K., Fasteen, J., Francis, A., Sullivan, E., & Wendt, T. (2020). Integrating Programming across the Undergraduate Mathematics Curriculum. *PRIMUS*, *30*(7), 735-749. https://doi.org/10.1080/10511970.2019.1616637

Cobb, P. (1994). Where Is the Mind? Constructivist and Sociocultural Perspectives on Mathematical Development. *Educational Researcher*, *23*(7), 13-20. https://doi.org/10.3102/0013189X023007013

Coles, A., & Sinclair, N. (2019). Re-Thinking 'Concrete to Abstract' in Mathematics Education: Towards the Use of Symbolically Structured Environments. *Canadian Journal of Science, Mathematics and Technology Education, 19*(4), 465-480. https://doi.org/10.1007/s42330-019-00068-4

Corry, L. (2015). *A Brief History of Numbers*. Oxford University Press.

Danish, J. A., & Gresalfi, M. (2018). Cognitive and Sociocultural Perspectives on Learning: Tensions and Synergy in the Learning Sciences. In F. Fischer, C. E. Hmelo-Silver, S. R. Goldman, & P. Reimann (Eds.), *International Handbook of the Learning Sciences* (pp. 34-43). Routledge. https://doi.org/10.4324/9781315617572

Denning, P. (2019). *Computational Thinking*. MIT Press.

Denning, P. J. (2017). Remaining Trouble Spots with Computational Thinking: Addressing Unresolved Questions Concerning Computational Thinking. *Communications of the ACM, 60*(6), 33-39. https://doi.org/10.1145/2998438

Denny, P., Luxton-Reilly, A., Tempero, E., & Hendrickx, J. (2011). Understanding the Syntax Barrier for Novices. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education, Acm* (pp. 208-212). https://doi.org/10.1145/1999747.1999807

Didiş Kabar, M. G. (2018). Secondary School Students' Conception of Quadratic Equations with One Unknown. *International Journal for Mathematics Teaching & Learning, 19*(1), 112-129.

Didiş, M. G., Baş, S., & Erbaş, A. K. (2011). Students' Reasoning in Quadratic Equations with One Unknown. In M. Pytlak, T. Rowland, & E. Swoboda (Eds.), *Proceedings of the Seventh Congress of the European Society for Research in Mathematics Education* (pp. 479-489). https://hal.archives-ouvertes.fr/hal-02158191

Didiş, M. G., & Erbas, A. K. (2015). Performance and Difficulties of Students in Formulating and Solving Quadratic Equations with One Unknown. *Educational Sciences: Theory & Practice, 15*(4), 1137-1150. https://doi.org/10.12738/estp.2015.4.2743

Dimos, I., Velaora, C., & Kakarountas, A. (2022). Computational Thinking in Greek Educational System for K-12: Towards the Future Teaching Approach. In *2022 Panhellenic Conference on Electronics & Telecommunications* (pp. 1-6). https://doi.org/10.1109/PACET56979.2022.9976359

Dohn, N. B. (2020). Students' Interest in Scratch Coding in Lower Secondary Mathematics. *British Journal of Educational Technology, 51*(1), 71-83. https://doi.org/10.1111/bjet.12759

Dong, Y., Marwan, S., Cateté, V., Price, T., & Barnes, T. (2019). Defining Tinkering Behavior in Open-Ended Block-Based Programming Assignments. In *Proceedings of the 50th Acm Technical Symposium on Computer Science Education* (pp. 1204-1210). https://doi.org/10.1145/3287324.3287437

Dreyfus, T., Hershkowitz, R., & Schwarz, B. (2015). The Nested Epistemic Actions Model for Abstraction in Context: Theory as Methodological Tool and Methodological Tool as Theory. In A. Bikner-Ashbahs, C. Knipping, & N. Presmeg (Eds.), *Approaches to Qualitative Research in Mathematics*

Education (pp. 185-217). Springer. https://doi.org/10.1007/978-94-017-9181-6_8

Dreyfus, T., & Kidron, I. (2014). Introduction to Abstraction in Context (Aic). In A. Bikner-Ahsbahs & S. Prediger (Eds.), *Networking of theories as a research practice in mathematics education* (pp. 85-96). Springer.

du Boulay, B. (1986). Some Difficulties of Learning to Program. *Journal of Educational Computing Research, 2*(1), 57-73.

Eckert, D., Timmermann, D., & Kautz, C. (2023). Student Misconceptions About Loops in Introductory Programming Courses and the Influence of Representations. In *IEEE Frontiers in Education Conference (Fie)* (pp. 1-5). https://doi.org/10.1109/FIE56618.2022.9962545

Edmonds, A. (1990). Factorising Quadratics. *Mathematics in School, 19*(3), 43. https://www.jstor.org/stable/30214685

Ely, R., & Adams, A. E. (2012). Unknown, Placeholder, or Variable: What Is X? *Mathematics Education Research Journal, 24*(1), 19-38. https://doi.org/https://doi.org/10.1007/s13394-011-0029-9

Ernest, P. (2019). Privilege, Power and Performativity: The Ethics of Mathematics in Society and Education. *Philosophy of Mathematics Education Journal*(35), 1-19.

Forsström, S. E., & Kaufmann, O. T. (2018). A Literature Review Exploring the Use of Programming in Mathematics Education. *International Journal of Learning, Teaching and Educational Research, 17*(12), 18-32. https://doi.org/10.26803/ijlter.17.12.2

Freudenthal, E. A., Lim, K. H., Ramirez, J. L., Kranz, S., & Tabor, C. (2013). Using Programming to Strengthen Mathematics Learning in 9th Grade Algebra Classes. In *2013 Asee Annual Conference and Exposition, Conference Proceedings*. https://doi.org/10.18260/1-2--22717

Freudenthal, H. (1973). *Mathematics as an Educational Task*. Springer Dordrecht.

Friberg, J. (2014). Geometric Division Problems, Quadratic Equations, and Recursive Geometric Algorithms in Mesopotamian Mathematics. *Archive for History of Exact Sciences, 68*(1), 1-34. https://doi.org/10.1007/s00407-013-0122-4

Fujiwara, K., Fushida, K., Tamada, H., Igaki, H., & Yoshida, N. (2012). Why Novice Programmers Fall into a Pitfall?: Coding Pattern Analysis in Programming Exercise. In *Fourth International Workshop on Empirical Software Engineering in Practice* (pp. 46-51). https://doi.org/10.1109/IWESEP.2012.13

Garner, S., Haden, P., & Robins, A. (2005). My Program Is Correct but It Doesn't Run: A Preliminary Investigation of Novice Programmers' Problems. In A. Young & D. Tolhurst (Eds.), *Australasian Computing Education Conference 2005. Conferences in Research and Practice in Information Technology.* (Vol. 42).

Germia, E., & Panorkou, N. (2020). Using Scratch Programming to Explore Coordinates. *Mathematics Teacher, 113*(4), 293-300. https://doi.org/10.5951/mtltpk12.2018.0032

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. In *IEEE International Conference on Engineering, Technology and Education* (pp. 1-7). https://doi.org/10.1109/TALE48000.2019.9225951

Gomes, C. A., Gomes, H., Figueiredo, M., Lucas, A., & Menezes, L. (2022). Mindmaths: Learning Mathematics in the Early Years through Computational Thinking and Robotics. In *2022 International Symposium on Computers in Education* (pp. 1-6). https://doi.org/10.1109/SIIE56031.2022.9982320

Gray, E. M., & Tall, D. O. (1994). Duality, Ambiguity, and Flexibility: A "Proceptual" View of Simple Arithmetic. *Journal for Research in Mathematics Education, 25*(2), 116-140. https://doi.org/10.2307/749505

Grover, S., & Basu, S. (2017). Measuring Student Learning in Introductory Block-Based Programming: Examining Misconceptions of Loops, Variables, and Boolean Logic. In *Proceedings of the 2017 Acdm Sigcse Technical Symposium on Computer Science Education* (pp. 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

Hansen, N. K., & Hadjerrouit, S. (2022). Analyzing Students' Computational Thinking and Programming Skills for Mathematical Problem Solving. In D. Ifenthaler, D. G. Sampson, & P. Isaías (Eds.), *Open and Inclusive Educational Practice in the Digital World* (pp. 155-173). Springer International Publishing. https://doi.org/10.1007/978-3-031-18512-0_10

Harel, G., & Tall, D. (1991). The General, the Abstract, and the Generic in Advanced Mathematics. *For the Learning of Mathematics, 11*(1), 38-42.

Heeffer, A. (2014). Was Uncle Tom Right That Quadratic Problems Can't Be Solved with the Rule of False Position? *The Mathematical Intelligencer, 36*(3), 65-69. https://doi.org/10.1007/s00283-013-9404-6

Hershkowitz, R., Dreyfus, T., Ben-Zvi, D., Friedlander, A., Hadas, N., Resnick, T., Tabach, M., & Schwarz, B. B. (2002). Mathematics Curriculum Development for Computerized Environments: A Designer-Researcher-Learner-Activity. In L. D. English (Ed.), *Handbook of International Research in Mathematics Education* (pp. 657-694). Lawrence Erlbaum.

Hershkowitz, R., Hadas, N., Dreyfus, T., & Schwarz, B. (2007). Abstracting Processes, from Individuals' Constructing of Knowledge to a Group's "Shared Knowledge". *Mathematics Education Research Journal, 19*(2), 41-68. https://doi.org/10.1007/BF03217455

Hershkowitz, R., Schwarz, B. B., & Dreyfus, T. (2001). Abstraction in Context: Epistemic Actions. *Journal for Research in Mathematics Education, 32*(2), 195-222. https://doi.org/10.2307/749673

Hillitt, G. (2007). Yet Another Method for Factorizing Quadratics! *Mathematics in School, 36*(2), 3-4. https://www.jstor.org/stable/30215998

Holmes, K., Prieto-Rodriguez, E., Hickmott, D., & Berger, N. (2018). Using Coding to Teach Mathematics: Results of a Pilot Project. In *Integrated education for the real world. 5th international STEM in education conference* (pp. 135-141). https://stem-in-ed2018.com.au/wp-content/uploads/2019/02/5th-International-STEM-in-Education-Post-Conference-Proceedings-2018.pdf

Hoyles, C., & Noss, R. (Eds.). (1992). *Learning Matematics and Logo*. MIT Press.

Hu, Q., Son, J.-W., & Hodge, L. (2021). Algebra Teachers' Interpretation and Responses to Student Errors in Solving Quadratic Equations. *International*

Journal of Science and Mathematics Education, 20(3). https://doi.org/10.1007/s10763-021-10166-1
Hungerbühler, N. (2020). An Alternative Quadratic Formula. *Mathematische Semesterberichte, 67*(1), 85-95. https://doi.org/10.1007/s00591-019-00262-3
Jahnke, A. (2020). *Programmering I Skolan: Vad, Hur, När Och Varför?* (2020:5). Ifous. https://www.ifous.se/app/uploads/2020/11/202010-ifous-2020-5-f.pdf
Kafai, Y. B., & Burke, Q. (2015). Constructionist Gaming: Understanding the Benefits of Making Games for Learning. *Educational Psychologist, 50*(4), 313-334. https://doi.org/10.1080/00461520.2015.1124022
Kalelioglu, F., Gulbahar, Y., & Kukul, V. (2016). A Framework for Computational Thinking Based on a Systematic Research Review. *Baltic Journal of Modern Computing, 4*(3), 583-596.
Kallia, M., van Borkulo, S. P., Drijvers, P., Barendsen, E., & Tolboom, J. (2021). Characterising Computational Thinking in Mathematics Education: A Literature-Informed Delphi Study. *Research in Mathematics Education, 23*(2), 159-187. https://doi.org/10.1080/14794802.2020.1852104
Kandemir, C. M., Kalelioğlu, F., & Gülbahar, Y. (2020). Pedagogy of Teaching Introductory Text-Based Programming in Terms of Computational Thinking Concepts and Practices. *Computer Applications in Engineering Education, 29*(1), 29-45. https://doi.org/10.1002/cae.22374
Karavakou, M., & Kynigos, C. (2020). Designing Periodic Logos: A Programming Approach to Understand Trigonometric Functions. In A. Donevska-Todorova, E. Faggiano, J. Trgalova, Z. Lavicza, R. Weinhandl, A. Clark-Wilson, & H.-G. Weigand (Eds.), *Proceedings of the Tenth ERME Topic Conference (ETC 10) on Mathematics Education in the Digital Age (MEDA)* (pp. 223-230). https://hal.archives-ouvertes.fr/hal-02932218
Katz, V. J. (2009). *A History of Mathematics: An Introduction* (3rd ed ed.). Addison-Wesley.
Katz, V. J., & Barton, B. (2007). Stages in the History of Algebra with Implications for Teaching. *Educational Studies in Mathematics, 66*(2), 185-201. https://doi.org/10.1007/s10649-006-9023-7
Katz, V. J., & Parshall, K. H. (2014). *Taming the Unknown: A History of Algebra from Antiquity to the Early Twentieth Century*. Princeton University Press.
Ke, F. (2014). An Implementation of Design-Based Learning through Creating Educational Computer Games: A Case Study on Mathematics Learning during Design and Computing. *Computers & Education, 73*, 26-39. https://doi.org/10.1016/j.compedu.2013.12.010
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
Kidron, I. (2008). Abstraction and Consolidation of the Limit Procept by Means of Instrumented Schemes: The Complementary Role of Three Different Frameworks. *Educational Studies in Mathematics, 69*(3), 197-216. https://doi.org/10.1007/s10649-008-9132-6
Kilhamn, C., Bråting, K., Helenius, O., & Mason, J. (2022). Variables in Early Algebra: Exploring Didactic Potentials in Programming Activities. *ZDM* –

Mathematics Education, 54(6), 1273-1288. https://doi.org/10.1007/s11858-022-01384-0

Kilhamn, C., Bråting, K., & Rolandsson, L. (2021). Teachers' Arguments for Including Programming in Mathematics Education. In G. A. Nortvedt, N. F. Buchholtz, J. Fauskanger, M. Hähkiöniemi, B. E. Jessen, M. Naalsund, H. K. Nilsen, G. Pálsdóttir, P. Portaankorva-Koivisto, J. Radišić, J. Ö. Sigurjónsson, O. Viirman, & A. Wernberg (Eds.), *Proceedings of Norma 20. The Ninth Nordic Conference on Mathematics Education.* (pp. 169-176).

Kilhamn, C., Ronaldsson, L., & Bråting, K. (2021). Programmering I Svensk Skolmatematik. *LUMAT General Issue, 9*(1), 283-312. https://doi.org/10.31129/ LUMAT.9.2.1457

Kohn, T. (2017). Variable Evaluation: An Exploration of Novice Programmers' Understanding and Common Misconceptions. In *Proceedings of the 2017 Acm Sigcse Technical Symposium on Computer Science Education* (pp. 345–350). Association for Computing Machinery. https://doi.org/https://doi.org/10.1145/3017680.3017724

Kopp, E. (2020). *Making up Numbers: A History of Invention in Mathematics*. Open Book Publishers. https://doi.org/https://doi.org/10.11647/OBP.0236

Kotsopoulos, D. (2007). Unravelling Student Challenges with Quadratics: A Cognitive Approach. *Australian Mathematics Teacher, 63*(2), 19-24.

Kouropatov, A., & Dreyfus, T. (2017). Combining Two Theories in the Design for Learning About Integrals. In T. Dooley, & Gueudet, G. (Ed.), *Proceedings of the Tenth Congress of the European Society for Research in Mathematics Education (CERME 10)* (pp. 2746-2753). DCU Institute of Education and ERME. https://hal.archives-ouvertes.fr/hal-01948861

Krieger, S., Allen, M., & Rawn, C. (2015). Are Females Disinclined to Tinker in Computer Science? In *Proceedings of the 46th Acm Technical Symposium on Computer Science Education* (pp. 102-107). https://doi.org/10.1145/2676723.2677296

Kučak, D., & Kučak, M. (2022). Gamification in Computer Programming Education – Systematic Literature Review. In *45th Jubilee International Convention on Information, Communication and Electronic Technology (Mipro)* (pp. 517-520). https://doi.org/10.23919/MIPRO55190.2022.9803457

Lahtinen, E., Ala-Mutka, K., & Järvinen, H.-M. (2005). A Study of the Difficulties of Novice Programmers. *ACM SIGCSE Bulletin, 37*(3), 14-18. https://doi.org/10.1145/1151954.1067453

Lambić, D. (2011). Presenting Practical Application of Mathematics by the Use of Programming Software with Easily Available Visual Components. *Teaching Mathematics and its Applications, 30*(1), 10-18. https://doi.org/10.1093/teamat/hrq014

Lemańska, M., Semanišinová, I., Calvo, C. S., Souto Salorio, M. J., & Tarrío Tobar, A. D. (2014). Geometrical Versus Analytical Approach in Problem Solving-an Exploratory Study. *Teaching of Mathematics, 17*(2), 84-95.

Lithner, J. (2008). A Research Framework for Creative and Imitative Reasoning. *Educational Studies in Mathematics, 67*(3), 255-276. https://doi.org/10.1007/s10649-007-9104-2

Lithner, J. (2017). Principles for Designing Mathematical Tasks That Enhance Imitative and Creative Reasoning. *ZDM, 49*(6), 937-949. https://doi.org/10.1007/s11858-017-0867-3

Liu, H., Wu, Z., Lu, Y., & Zhu, L. (2023). Exploring the Balance between Computational Thinking and Learning Motivation in Elementary Programming Education: An Empirical Study with Game-Based Learning. *IEEE Transactions on Games, 15*(1), 95-107. https://doi.org/10.1109/TG.2022.3143701

Lobato, J., Ellis, A. B., & Muñoz, R. (2003). How "Focusing Phenomena" in the Instructional Environment Support Individual Students' Generalizations. *Mathematical Thinking and Learning, 5*(1), 1-36. https://doi.org/10.1207/S15327833MTL0501_01

Loh, P. S. (2019). *A Simple Proof of the Quadratic Formula.* arXiv. https://doi.org/10.48550/arXiv.1910.06709

Lokkila, E., Christopoulos, A., & Laakso, M.-J. (2023). A Data-Driven Approach to Compare the Syntactic Difficulty of Programming Languages. *Journal of Information Systems Education, 34*(1), 84-93. https://jise.org/Volume34/n1/JISE2023v34n1pp84-93.html

López, J., Robles, I., & Martínez-Planell, R. (2016). Students' Understanding of Quadratic Equations. *International Journal of Mathematical Education in Science and Technology, 47*(4), 552-572. https://doi.org/10.1080/0020739X.2015.1119895

Lovric, M. (2018). Programming and Mathematics in an Upper-Level University Problem-Solving Course. *PRIMUS, 28*(7), 683-698. https://doi.org/10.1080/10511970.2017.1403524

Lundin, S. (2017). How Mathematics Education Became a Ritual. In U. Kortenkamp & A. Kuzle (Eds.), *Beiträge Zum Mathematikunterricht 2017. (51. Jahrestagung Der Gesellschaft Für Didaktik Der Mathematik)* (pp. 19-26). WTM-Verlag.

Luxton-Reilly, A. (2016). Learning to Program Is Easy. In *Proceedings of the 2016 Acm Conference on Innovation and Technology in Computer Science Education*. 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). Inroductory Programming: A Systematic Literature Review. In *Proceedings of the 23rd Annual Acm Conference on Innovation and Technology in Computer Science Education* (pp. 55-106). https://doi.org/10.1145/3293881.3295779

Lyon, L. A., Clayton, C., & Green, E. (2018). Tinkering in the Wild: What Leads to Success for Female End-User Programmers? In *IEEE Symposium on Visual Languages and Human-Centric Computing (Vl/Hcc)* (pp. 125-129). https://doi.org/10.1109/VLHCC.2018.8506561

MacGregor, M., & Stacey, K. (1997). Students' Understanding of Algebraic Notation: 11-15. *Educational Studies in Mathematics, 33*(1), 1-19.

Maher, C. A., & Sigley, R. (2020). Task-Based Interviews in Mathematics Education. In S. Lerman (Ed.), *Encyclopedia of Mathematics Education* (pp. 821-824). Springer International Publishing. https://doi.org/10.1007/978-3-030-15789-0_147

Makonye, J., & Nhlanhla, S. (2014). Exploring 'Non-Science' Grade 11 Learners' Errors in Solving Quadratic Equations. *Mediterranean Journal of Social Sciences, 5*(27), 634-644. https://doi.org/10.5901/mjss.2014.v5n27p634

Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., & Settle, A. (2014). Computational Thinking in K-9 Education. In *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference* (pp. 1-29). https://doi.org/10.1145/2713609.2713610

Maryati, T. K., Aisah, K., & Miftah, R. (2021). How the Asian and European Communities Accept Negative Numbers. *Journal of Education in Muslim Society, 8*(2), 167-178.

Memnun, D. S., Aydin, B., Dinç, E., Çoban, M., & Sevindik, F. (2015). Failures and Inabilities of High School Students About Quadratic Equations and Functions. *Journal of Education and Training Studies, 3*(6), 50-60. https://doi.org/10.11114/jets.v3i6.918

Misfeldt, M., & Ejsing-Duun, S. (2015). Learning Mathematics through Programming: An Instrumental Approach to Potentials and Pitfalls. In K. Krainer & N. Vondrová (Eds.), *Proceedings of the Ninth Congress of the European Society for Research in Mathematics Education* (pp. 2524–2530). Charles University in Prague.

Nahin, P. J. (2010). *An Imaginary Tale: The Story of √-1*. Pinceton University Press.

Niemelä, P., Partanen, T., Harsu, M., Leppänen, L., & Ihantola, P. (2017). Computational Thinking as an Emergent Learning Trajectory of Mathematics. In *Proceedings of the 17th Koli Calling International Conference on Computing Education Research* (pp. 70-79). ACM.

Niss, M. (1994). Mathematics in Society. In R. Biehler, R. W. Scholz, R. Sträßer, & B. Winkelmann (Eds.), *Didactics of Mathematics as a Scientific Discipline* (pp. 367-378). Kluwer Academic Publishers.

Noss, R., & Hoyles, C. (1996). *Windows on Mathematical Meanings : Learning Cultures and Computers*. Kluwer Academic Publishers.

Nurhasanah, F., Kusumah, Y. S., Sabandar, J., & Suryadi, D. (2017). Mathematical Abstraction: Constructing Concept of Parallel Coordinates. *Journal of Physics: Conference Series, 895*(1), 1-8. https://doi.org/10.1088/1742-6596/895/1/012076

Nurhasanah, F., & Sumekar, W. (2019). Context Design for Analyzing Students' Abstraction with Different Cognitive Styles in Learning Tangent Lines of Circles. In N. Y. Indriyanti, M. Ramli, & F. Nurhasanah (Eds.), *The 2nd International Conference Oon Science, Mathematics, Environment, and Education* (pp. 1-8). https://doi.org/10.1063/1.5139806

O'Connor, B. R., & Norton, S. (2016). Investigating Students' Mathematical Difficulties with Quadratic Equations. In B. White, M. Chinnappan, & S. Trenholm (Eds.), *Opening up Mathematics Education Research: Proceedings of the 39th Annual Conference of the Mathematics Education Research Group of Australasia.* (pp. 1-8). MERGA.

Olteanu, C. (2022). Programming, Mathematical Reasoning and Sense-Making. *International Journal of Mathematical Education in Science and Technology, 53*(8), 2046-2064. https://doi.org/10.1080/0020739X.2020.1858199

Olteanu, C., & Holmqvist, M. (2012). Differences in Success in Solving Second-Degree Equations Due to Differences in Classroom Instruction. *International Journal of Mathematical Education in Science and Technology, 43*(5), 575-587. https://doi.org/10.1080/0020739X.2011.622807

Oxford. (2016). *A Dictionary of Computer Science* (A. Buterfield, G. E. Ngondi, & A. Kerr, Eds. 7 ed.). Oxford University Press.

Ozmantar, M. F. (2004). Scaffolding, Abstraction, and Emergent Goals. *Proceedings of the British Society for Research into Learning Mathematics, 24*(2), 83-89.

Pantsar, M. (2024). Theorem Proving in Artificial Neural Networks: New Frontiers in Mathematical Ai. *European Journal for Philosophy of Science, 14*(1), 1-22. https://doi.org/10.1007/s13194-024-00569-6

Paparistodemou, E., Meletiou-Mavrotheris, M., & Vasou, C. (2017). *Insights from Students' Reasoning About Probability When They Design Their Own Scratch Games* CERME 10, Dublin, Ireland. https://hal.archives-ouvertes.fr/hal-01927781

Papert, S. (1980). *Mindstorms : Children, Computers, and Powerful Ideas*. Basic books.

Papert, S. (1996). An Exploration in the Space of Mathematics Educations. *International Journal of Computers for Mathematical Learning, 1*(1), 95-123.

Pea, R. D. (1986). Language-Independent Conceptual "Bugs" in Novice Programming. *Journal of Educational Computing Research, 2*(1), 25-36.

Perkins, D. N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R. (1986). Conditions of Learning in Novice Programmers. *Journal of Educational Computing Research, 2*(1), 37-55. https://doi.org/10.2190/GUJT-JCBJ-Q6QU-Q9PL

Pólya, G. (1945). *How to Solve It*. Princeton University Press.

Prabawa, H. W., Rosjanuardi, R., & Nurlaelah, E. (2023). Problem Decomposition Skills, Mathematical Maturity, and Their Relation to Mathematics Problem-Solving in a Computer Science Learning Class. *Jurnal Kependidikan: Jurnal Hasil Penelitian dan Kajian Kepustakaan di Bidang Pendidikan, Pengajaran dan Pembelajaran, 9*(3), 946-958. https://doi.org/10.33394/jk.v9i3.8258

Prayogo, Suwarsono, & Khabibah, S. (2020). An Illustration of Mathematical Skills: The Procept (Processconcept) Junior High School Students in Solving Mathematical Problems. *Journal of Physics: Conference Series, 2157*, 1-8. https://doi.org/10.1088/1742-6596/2157/1/012039

Psycharis, S., & Kallia, M. (2017). The Effects of Computer Programming on High School Students' Reasoning Skills and Mathematical Self-Efficacy and Problem Solving. *Instructional Science, 45*(5), 583-602. https://doi.org/10.1007/s11251-017-9421-5

Putnam, R. T., Sleeman, D., Baxter, J. A., & Kuspa, L. K. (1986). A Summary of Misconceptions of High School Basic Programmers. *Journal of Educational Computing Research, 2*(4), 459-472.

Qian, Y., & Lehman, J. (2018). Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Transactions on Computing Education, 18*(1), 1-24. https://doi.org/10.1145/3077618

Radford, L. (2006). Algebraic Thinking and the Generalization of Patterns: A Semiotic Perspective. In S. Alatorre, J. L. Cortina, M. Sáiz, & A. Méndez (Eds.), *Proceedings of the 28th Conference of the International Group for the Psychology of Mathematics Education, North American Chapter* (Vol. 1, pp. 2-21).

Raptopoulou, A. (2021). *Politics of Contemporary Education Policy: The Case of Programming in the Swedish Curriculum* [Doctoral thesis, Stockholm University]. http://urn.kb.se/resolve?urn=urn:nbn:se:su:diva-195028

Rich, K. M., & Yadav, A. (2020). Applying Levels of Abstraction to Mathematics Word Problems. *TechTrends, 64*(3), 395-403. https://doi.org/10.1007/s11528-020-00479-3

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education, 13*(2), 137-172. https://doi.org/10.1076/csed.13.2.137.14200

Rodríguez-Martínez, J. A., González-Calero, J. A., & Sáez-López, J. M. (2020). Computational Thinking and Mathematics Using Scratch: An Experiment with Sixth-Grade Students. *Interactive Learning Environments, 28*(3), 316-327. https://doi.org/10.1080/10494820.2019.1612448

Sangwin, C. J., & O'Toole, C. (2017). Computer Programming in the Uk Undergraduate Mathematics Curriculum. *International Journal of Mathematical Education in Science and Technology, 48*(8), 1133-1152. https://doi.org/10.1080/0020739X.2017.1315186

Schanzer, E., Fisler, K., & Krishnamurthi, S. (2018). *Assessing Bootstrap: Algebra Students on Scaffolded and Unscaffolded Word Problems.* SIGCSE, Baltimore, Maryland, USA.

Schanzer, E. T. (2015). *Algebraic Functions, Computer Programming, and the Challenge of Transfer* [Doctoral Dissertation, Harvard]. Harvard Graduate School of Education repository. http://nrs.harvard.edu/urn-3:HUL.InstRepos:16461037

Scheiner, T. (2016). New Light on Old Horizon: Constructing Mathematical Concepts, Underlying Abstraction Processes, and Sense Making Strategies. *Educational Studies in Mathematics, 91*(2), 165-183. https://doi.org/10.1007/s10649-015-9665-4

Schoenfeld, A. (1988). When Good Teaching Leads to Bad Results: The Disasters of "Well-Taught" Mathematics Courses. *Educational Psychologist, 23*(2), 145-166.

Schwarz, B. B., Dreyfus, T., & Hershkowitz, R. (2009). The Nested Epistemic Actions Model for Abstraction in Context. In B. B. Schwarz, T. Dreyfus, & R. Hershkowitz (Eds.), *Transformation of Knowledge through Classroom Interaction* (pp. 11-41). Routledge.

Sfard, A. (1998). On Two Metaphors for Learning and the Dangers of Choosing Just One. *Educational Researcher, 27*(2), 4-13. https://doi.org/10.3102/0013189X027002004

Sim, T. Y., & Lau, S. L. (2022). Review on Challenges and Solutions in Novice Programming Education. In *IEEE International Conference on Computing* (pp. 55-61). https://doi.org/10.1109/ICOCO56118.2022.10031657

Skolverket. (2018a). *Curriculum for the Compulsory School, Preschool Class and School-Age Educare (Revised 2018).* https://www.skolverket.se/publikationer?id=3984

Skolverket. (2018b). *Matematik*.
Skolverket. (2022a). *Läroplan För Grundskolan, Förskoleklassen Och Fritidshemmet – Lgr22*. https://www.skolverket.se/publikationer?id=9718
Skolverket. (2022b). *Matematik*. https://www.skolverket.se/undervisning/gymnasieskolan/laroplan-program-och-amnen-i-gymnasieskolan/gymnasieprogrammen/amne?url=-996270488%2Fsyllabuscw%2Fjsp%2Fsubject.htm%3FsubjectCode%3DMAT%26courseCode%3DMATMAT01a%26version%3D11%26tos%3Dgy&sv.url=12.5dfee44715d35a5cdfa92a3#anchor_MATMAT01a
Sleeman, D., Putnam, R. T., Baxter, J., & Kuspa, L. (1986). Pascal and High School Students: A Study of Errors. *Journal of Educational Computing Research, 2*(1), 5-23.
So, M., & Kim, J. (2018). An Analysis of the Difficulties of Elementary School Students in Python Programming Learning. *International Journal on Advanced Science, Engineering and Information Technology, 8*(4-2), 1507-1512. https://doi.org/10.18517/ijaseit.8.4-2.2720
Stehlikova, N. (2003). Building a Finite Arithmetic Structure: Interpretation in Terms of Abstraction in Context. In M. A. Mariotti (Ed.), *Proceedings of the Third Conference of the European Society for Research in Mathematics Education* (pp. 1-10).
Stigberg, H., & Stigberg, S. (2020). Teaching Programming and Mathematics in Practice: A Case Study from a Swedish Primary School. *Policy Futures in Education, 18*(4), 483-496. https://doi.org/10.1177/1478210319894785
Tabach, M., Rasmussen, C., Dreyfus, T., & Hershkowitz, R. (2017). Abstraction in Context and Documenting Collective Activity. In T. Dooley & G. Gueudet (Eds.), *Proceedings of the Tenth Congress of the European Society for Research in Mathematics Education* (pp. 2692-2699).
Tall, D. (1992). The Transition from Arithmetic to Algebra: Number Patterns, or Proceptual Programming. *New Directions in Algebra Education, 9*, 213-231.
Tall, D., de Lima, R. N., & Healy, L. (2014). Evolving a Three-World Framework for Solving Algebraic Equations in the Light of What a Student Has Met Before. *The Journal of Mathematical Behavior, 34*, 1-13. https://doi.org/10.1016/j.jmathb.2013.12.003
Tamborg, A. L. (2022). A Solution to What? Aims and Means of Implementing Informatics-Related Subjects in Denmark, Sweden, and England. *Acta Didactica Norden, 16*(4), 1-26. https://doi.org/10.5617/adno.9184
Tan, P. H., Ting, C. Y., & Ling, S. W. (2009). Learning Difficulties in Programming Courses Undergraduate Perspective and Perception. In *International Conference on Computer Technology and Development* (pp. 2-46). https://doi.org/10.1109/ICCTD.2009.188
Tedre, M., & Denning, P. J. (2016). The Long Quest for Computational Thinking. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research* (pp. 120-129). https://doi.org/10.1145/2999541.2999542
Timotheou, S., & Ioannou, A. (2019). On Making, Tinkering, Coding and Play for Learning: A Review of Current Research. In *17th Ifip Conference on Human-Computer Interaction (Interact)* (pp. 217-232). https://doi.org/10.1007/978-3-030-29384-0_14

Treffers, A. (1987). *Three Dimensions. A Model of Goal and Theory Description in Mathematics Education: The Wiskobas Project.* Springer Dordrecht. https://doi.org/https://doi.org/10.1007/978-94-009-3707-9

Vaiyavutjamai, P., & Clements, M. A. (2006). Effects of Classroom Instruction on Students' Understanding of Quadratic Equations. *Mathematics Education Research Journal, 18*(1), 47-77. https://doi.org/10.1007/BF03217429

Vetenskapsrådet. (2017). *God Forskningssed*. https://www.vr.se/download/18.2412c5311624176023d25b05/1555332112063/God-forskningssed_VR_2017.pdf

Vinnervik, P. (2022). Implementing Programming in School Mathematics and Technology: Teachers' Intrinsic and Extrinsic Challenges. *International Journal of Technology and Design Education, 32*(1), 213-242. https://doi.org/10.1007/s10798-020-09602-0

Vinnervik, P., & Bungum, B. (2022). Computational Thinking as Part of Compulsory Education: How Is It Represented in Swedish and Norwegian Curricula? *Nordic Studies in Science Education, 18*(3), 384-400. https://doi.org/10.5617/nordina.9296

Wang, X. q., Qi, C. y., & Wang, K. (2017). A Categorization Model for Educational Values of the History of Mathematics: An Empirical Study. *Science & Education, 26*(7-9), 1029-1052. https://doi.org/10.1007/s11191-017-9937-8

Watson, C., & Li, F. (2014). Failure Rates in Introductory Programming Revisited. In *Proceedings of the 2014 Conference on Innovation Technology in Computer Science Education* (pp. 39-44). https://doi.org/10.1145/2591708.2591749

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining Computational Thinking for Mathematics and Science Classrooms. *Journal of Science Education and Technology, 25*(1), 127-147. https://doi.org/10.1007/s10956-015-9581-5

Wing, J. M. (2006). Computational Thinking. *Communications of the ACM, 49*(3), 33-35.

Wing, J. M. (2010). *Computational Thinking: What and Why?* http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf

Winslow, L. E. (1996). Programming Pedagogy—a Psychological Overview. *ACM SIGCSE Bulletin, 28*(3), 17-22. https://doi.org/10.1145/234867.234872

Yahya, N., & Shahrill, M. (2015). The Strategies Used in Solving Algebra by Secondary School Repeating Students. *Procedia - Social and Behavioral Sciences, 186*, 1192-1200. https://doi.org/10.1016/j.sbspro.2015.04.168

Žanko, Ž., Mladenović, M., & Krpan, D. (2020). Analysis of School Students' Misconceptions About Basic Programming Concepts. *Journal of Computer Assisted Learning, 38*(3), 719-730. https://doi.org/10.1111/jcal.12643

Öztürk, F. U., Özdemir, M., & Ozbasi, D. (2021). Investigation of Project-Based Learning Method in Teaching Programming in Terms of Academic Achievement, Cognitive Load and Behavior Change. *Educational Policy Analysis and Strategic Research, 16*(2), 276-294. https://doi.org/10.29329/epasr.2020.345.12

Appendix 1 Programming Tasks

This appendix contains the programming tasks given to the students in the different interventions. Note that all of the tasks and code examples have been translated into English from Swedish.

A1.1 Tasks Used in Intervention 1

Below are descriptions of the tasks used in the first intervention. Each task starts with an summary of the intended goals and purpose of the task. Each task then contains the following sections:

- **Predict**. This text was given to the students as an initial pen and paper activity. The intention was to help lead their thinking in the right direction before focusing on the technical aspects of programming.
- **Code**. This text was given to the students when they were done with the paper and pencil activity and contains instructions for the programming task.
- **Example output**. This section was for the teacher only and shows examples of the type of output expected from the students. The intention here was only to provide examples and not to list the exact output expected.
- **Example Python program**. This section was also only for the teacher and contains an example solution to the programming task written in Python. As with the example output, the intention here was only to provide one example of how the task might be completed and not the show the expected or only correct solution.

A1.1.1 Task Overview

The following is a brief overview of the tasks:

1. Write a program that asks the user to input an integer. The program should then output pairs of numbers such that their product is 0 and their difference is the integer that the user entered.
2. Modify your previously written program to also output three different equations representing the situation based on the user's input.
3. Write a program that asks the user to input two integer solutions to a quadratic equation. The solutions can be positive or negative. The program should then output a quadratic equation in factored form that has those solutions as well as in standard form.
4. Write a program that asks the user to input an integer. The program should then output all possible factor pairs for that number. Next to each factor pair, include their sum. `Number: 6 Factors: (1,6)  Sum: 1 + 6 = 7`
5. Consider the following quadratic equation in standard form: $x^2 + bx + c = 0$. Write a program that outputs that equation with all integer combinations of b and c between -5 and 5. For example: $x^2 - 5x - 5 = 0, x^2 - 5x - 4 = 0, \dots, x^2 + x + 2 = 0, \dots, x^2 + 5x + 5 = 0$

   The program should then try and factor the associated quadratic equation. If the equation is factorable, the program should output the equation in standard form and factored form. For example: $x^2 - 5x + 4 = (x - 1)(x - 4) = 0$
6. Modify your previous program to also print out the solutions to the equations which have integer solutions. The program should show the calculation verifying the solutions using the factored form.

A1.1.2 Task 1: Two Constrained Numbers Whose Product Is 0

Problems addressed	Misunderstanding the zero product property.
Goal	This task is designed to construct an understanding of the zero product property as well as laying some groundwork for connecting it to quadratic equations.
Procepts	Both the concept and the concrete implementation of the zero product property
Programming benefits	Writing the program should focus the students' attention on the details of the zero product property. Printing two solutions should lay the groundwork for understanding that there are always two solutions to quadratic equations. Realizing that no special code is necessary for handling negative numbers should help ease work with negative numbers.

Predict

The product of two numbers is 0. The difference of the numbers is 2. What can you say about the numbers? How would this be different if the product were 4 instead of 0?

Code

Write a program that asks the user to input an integer. The program should then output pairs of numbers such that their product is 0 and their difference is the integer that the user entered. The program should show the full calculation confirming that the product is 0 for each pair of numbers.

Example output

```
>>>
Enter an integer: 4
0 * 4 = 0
-4 * 0 = 0
>>>
Enter an integer: -2
0 * -2 = 0
2 * 0 = 0
```

Example Python program

```
num = int(input("Enter an integer: "))
print(f"0 * {num} = 0")
print(f"{-num} * 0 = 0")
```

A1.1.3 Task 2: Generate Equations for the Previous Task

Problems addressed	Misunderstanding the zero product property. Not understanding that both *x*'s in a factored equation must have the same value. Not understanding that there are two solutions to a quadratic equation.
Goal	This task is designed to deepen the understanding of the zero product property as well as further connecting it to quadratic equations and that there are always two solutions.
Procepts	Both the concept of and the concrete implementation of the zero product property. Both the object of a factored quadratic equation as well as how to evaluate it for values of *x*.
Programming benefits	Writing the program should focus the students' attention on the fact that both *x*'s are the same variable and must have the same value when evaluating the factored expression.

Predict

Return to the question explored in the previous task: The product of two numbers is 0. The difference of the numbers is 2. Can you write an equation with only one variable to represent this situation? Can you write two additional equations to represent this situation? How do the possible values of your variable in each equation differ from the numbers you found previously?

Code

Modify your previously written program to also output three different equations representing the situation based on the user's input. Your program should also output the possible values of the variables in the equations.

Example output

```
>>>
Enter an integer: 4
0 * 4 = 0
-4 * 0 = 0
x(x+4)=0
x=0 or x=4
x(x-4)=0
x=0 or x=-4
(x+1)(x+5)=0
x=-1 or x=-5
```

Example Python program

```
print(f"x(x-{num})")
print(f"x=0 or x={num}")
print(f"x(x+{num})")
print(f"x=0 or x={-num}")
print(f"(x+1)(x+{num+1})")
print(f"x=-1 or x={-(num+1)}")
```

A1.1.4 Task 3: A Quadratic Equation From Two Solutions

Problems addressed	Misunderstanding the zero product property. Not understanding the connection between the solutions and the equation in factored form. Not understanding the connection between the equation in standard form and in factored form.
Goal	This task is designed to allow the students to discover the relationship between the solutions to a quadratic equation and the coefficients when in standard form. This lays important groundwork for factoring quadratic equations as well as for understanding Vieta's formulas.
Procepts	Both the object of a factored quadratic equation as well as how to manipulate it algebraically. Both the concept of the connection between the solutions to a quadratic equation and its factored form as well as how to implement it.
Programming benefits	Writing the program should help students deepen their understanding of the connection between the solutions to a quadratic equation and the equation in factored form. It should also help students draw the connection between the solutions and the coefficients of the equation in standard form.

Predict

The solutions to a quadratic equation are $x = 3$ and $x = 4$. How would you go about writing a quadratic equation in factored form that has those two solutions? How would you then write the same equation in standard form? Can you use the solutions to help you with both forms?

Code

Write a program that asks the user to input two integer solutions to a quadratic equation. The solutions can be positive or negative. The program should then output a quadratic equation in factored form that has those two solutions. The program should then output the same equation in standard form.

Example output

```
>>>
Enter two integer solutions to a quadratic equation.
Solution 1 = 3
Solution 2 = 4
(x - 3)(x - 4) = 0
x² - 7x + 12 = 0
```

Example Python program

```
print("Enter two integer solutions to a quadratic equation.")
sol1 = int(input("Solution 1 = "))
sol2 = int (input("Solution 2 = "))
print(f"(x - ({sol1}))(x - ({sol2})) = 0")
b = -(sol1+sol2)
c = sol1 * sol2
print(f"x\u00B2 + ({b})x + ({c}) = 0")
```

A1.1.5 Task 4: Find All Signed Factor Pairs of an Integer

Problems addressed	Not understanding the connection between the equation in standard form and in factored form.
Goal	This task lays the groundwork for the connection between factoring the constant term and factoring the equation. This builds on the previous task where the students should have discovered that the constant term is the product of the solutions. This task also helps students understand the connection between standard form and factored form as well as the connection between the solutions and factored form.
Procepts	Both the concept of the connection between the solutions to a quadratic equation and its standard form as well as how to work with that connection numerically.
Programming benefits	Writing the program should help students deepen their understanding of the connection between the solutions to a quadratic equation and the coefficients of the equation in standard form.

Predict

Find all factor pairs of 6. For example: $2 \cdot 3 = 6$ and $-1 \cdot -6 = 6$. Next to each factor pair, include their sum. For example, $2 \cdot 3 = 6 \quad 2 + 3 = 5$. Is there a systematic way to find all the factor pairs? What would be different if the number was -12 instead of 6?

Code

Write a program that asks the user to input an integer. The program should then output all possible factor pairs for that number. Next to each factor pair, include their sum.

Example output

```
>>>
Enter an integer other than zero: 6
6 = 1 * 6   1 + 6 = 7
6 = -1 * -6   -1 + -6 = -7
6 = 2 * 3   2 + 3 = 5
6 = -2 * -3   -2 + -3 = -5
```

Example Python program

```
num = 0
while num == 0:
    num = int(input("Enter an integer other than zero: "))
max_check = int(abs(num)**0.5)+1
for d1 in range(1,max_check):
    if num % d1 == 0:
        d2 = num//divisor
        print(f"{num} = {d1} * {d2}   {d1} + {d2} = {d1+d2}")
        print(f"{num} = {-d1} * {-d2}   {-d1} + {-d2} = {-d1-d2}")
```

A1.1.6 Task 5: Factor a Quadratic Equation With Integer Solutions

Problems addressed	Not understanding the connection between the equation in standard form and in factored form. Not understanding the connection between the solutions to a quadratic equation and the equation in factored form.
Goal	This task builds on the previous two tasks and allows students to apply the understanding gained there to actually factor a quadratic equation when possible.
Procepts	Both the concept of the connection between the solutions to a quadratic equation and its standard form as well as how to work with that connection numerically.
Programming benefits	Writing the program should help students deepen their understanding of the connection between the solutions to a quadratic equation and the coefficients of the equation in standard form. In particular, in order to create an algorithm that can factor an equation over the integers, the students will need to have a deeper understanding of why the algorithm works.

Predict

How can you use the results of the previous activity to help you factor $x^2 + 5x + 6 = 0$? How can you use the results of the last activity to design a program to factor a quadratic equation with integer solutions?

Code

Consider the following quadratic equation in standard form: $x^2 + bx + c = 0$

Write a program generates all integer combinations of b and c between -5 and 5. The program should then try and factor the associated quadratic equation. If the equation if factorable, the program should output the equation in standard form and factored form.

Example output

```
>>>
x² - 5x + 4 = (x - 1)(x - 4) = 0
x² - 4x - 5 = (x + 1)(x - 5) = 0
x² - 4x + 3 = (x - 1)(x - 3) = 0
x² - 4x + 4 = (x - 2)(x - 2) = 0
x² - 3x - 4 = (x + 1)(x - 4) = 0
x² - 3x + 2 = (x - 1)(x - 2) = 0
x² - 2x - 3 = (x + 1)(x - 3) = 0
x² - 2x + 1 = (x - 1)(x - 1) = 0
x² - 1x - 2 = (x + 1)(x - 2) = 0
x² - 4 = (x + 2)(x - 2) = 0
```

Example Python program

```
for b in range(-5,6):
    for c in range(-5,6):
        for div1 in range(1, int(abs(c)**0.5)+1):
            if c % div1 == 0:
                div2=c//div1
                if div1+div2==b:
                    nicePrint(b,c,div1,div2)
                if div1+div2==-b and b!=0:
                    nicePrint(b,c,-div1,-div2)
```

A1.1.7 Task 6: Solve a Quadratic Equation With Integer Solutions

Problems addressed	Not understanding the connection between the equation in standard form and in factored form. Not understanding the connection between the solutions to a quadratic equation and the equation in factored form.
Goal	This task combines the understanding gained in all the previous tasks to solve a quadratic equation with integer solutions by factoring and using the zero product property.
Procepts	Both the concept of the connection between the solutions to a quadratic equation and its standard form as well as how to work with that connection numerically.
Programming benefits	Writing the program should help students deepen their understanding of the connection between the solutions to a quadratic equation and the coefficients of the equation in standard form. In particular, in order to create an algorithm that can factor an equation over the integers, the students will need to have a deeper understanding of why the algorithm works. Implementing a method for finding both solutions from factored form should help deepen this connection as well.

Predict

How can you use the results of the previous activity to help you solve $x^2 - 7x + 6 = 0$? How can you use the results of the last activity to design a program find the solutions to a quadratic equation that has integer solutions?

Code

Modify your previous program to also print out the solutions to the equations which have integer solutions. The program should show the calculation verifying the solutions using the factored form.

Example output

```
>>>
Enter integer coefficents for the quadratic equation:
x² + bx + c = 0
b = -5
c = 6
x² + -5x + 6 = 0
(x - 2)(x - 3) = 0
Solutions are: x = 2 or x = 3
```

Example Python program

```
for b in range(-5,6):
    for c in range(-5,6):
        for div1 in range(1, int(abs(c)**0.5)+1):
            if c % div1 == 0:
                div2=c//div1
                if div1+div2==b:
                    nicePrint(b,c,div1,div2)
                if div1+div2==-b and b!=0:
                    nicePrint(b,c,-div1,-div2)
```

A1.2 Tasks Used in Intervention 2

Below are descriptions of the tasks that were intended to be used in the second intervention. As with the tasks used in the first intervention, most of the information provided here is intended for the teacher.

A1.2.1 Changes Made to Tasks After Intervention 1

Tasks 1 and 2 were completely removed as they didn't seem to embody any significant task design principles in practice. In addition, there appeared to be a risk that students might have their attention drawn to the wrong aspects of the exercise, in which case programming would be acting as a detrimental focusing phenomenon (see section 9.3).

Task 4 (factoring an arbitrary integer) was changed to be the first task for a number of reasons, the main one being that all the rest of the tasks were more connected to each other than this one, even if it is essential for some later tasks. While an important part of reaching the final goals, the actual mathematical content of this task was not directly connected to rest of the mathematical content in a way that was obvious to the students. Having this task later and disrupting the flow made it harder for this task to meet the following task design criteria (see section 9.3):

- Encourage creativity and curiosity.
    - When the task felt "out of phase" with the rest of the mathematics, there was noticeable reduction in student interest and creativity when working with this task.
- Meaningful to the students.
    - When the task felt out of place, it also felt less meaningful to the students.

Since I had reason to believe that these students would be less proficient at programming, some changes were made in an attempt to simplify the programming requirements. This included dividing tasks up into smaller tasks, providing help code to get them started, and more clear guidelines for the teacher conducting the intervention. These changes were made in attempt to be more in line with the following task design criteria:

- The teacher has an important role in leading discussions and helping students focus on the process and important ideas in the inquiry and not just on the result.
- Lead the students to results which build their intuition and make them feel good.
- Don't overcomplicate tasks.
- Engage knowledge that students already have ... but challenges them to think harder or differently about what they know.
- This also made it easier for programming to act as a constructive focusing phenomenon and draw the students´ attention to the critical aspects of the task.

A1.2.2 Task Structure

- **Predict**. Some tasks start with one or more discussion questions which should be given to the students as pen and paper exercises. They are designed to highlight a few key elements and ideas that will help with the programming tasks. Note that not all tasks have a predict step.
- **Code**. This is the actual programming task that should be given to the students after they have thought about any the questions in the predict step.
- **Example output**. Example output provided for the teacher. Note that these are only examples of the type of output expected from the students. The intention here was only to provide examples and not to list the exact output expected.
- **Example Python program**. This section is also only for the teacher and contains an example solution to the programming task written in Python. As with the example output, the intention here was only to provide one example of how the task might be completed and not the show the expected or only correct solution.
- **Notes**. Most tasks also include detailed notes for the teacher. These notes cover previously seen problems students ran into when completing the task, important functionality to look out for in the students' programs, and technical advice, including some of the technical aspects that can just be explained or given to the students without compromising the intervention.

- **Skeleton code**. Some tasks provide explicit examples of code outlines that the teacher could provide to the students ahead of time to help with some of the technical aspects of writing the programs without compromising the intention of the intervention. These examples are written Python, though could be converted to any other text-based programming language which the students may prefer to work with.

A1.2.3 When Should the Teacher Help the Students?

This is perhaps one of the more difficult decisions to make during the intervention, especially since it has a tendency to come up repeatedly. The following are some guidelines that were developed before the second intervention.

Help them with coding

Unless otherwise stated in the task, it is in general okay, and even advisable, to help the students with technical issues related to programming. Examples include issues of syntax, suggestions on when to make something a function (compared to having it in the main body of code), advice on input and output methods (except where noted in the task), how to write to a file, how to save information to a list and then print the list, how to use nested loops, how to check for divisibility, etc.

Help them with unrelated mathematics

Questions involving mathematics not directly related to solving quadratic equations or factoring integers can be explained to the students without interfering with the intervention results.

Encourage experimentation

If the students get stuck on a piece of relevant mathematics, instead of explaining it to them, try and encourage them to experiment and test the results of their experiments. If at all possible, the important pieces of mathematics should not be explained to them as it may invalidate some of the intervention results. That being said, if they are making absolutely no progress and nothing else is working, clues may be given to help them move forward, at the teacher's discretion.

Provide general information about quadratic equations ahead of time

When appropriate during the tasks, or before they are started, the students should be presented with the following information about quadratic equations:

- The following is called *standard form* for a quadratic equations: $ax^2 + bx + c = 0$.
- They will only be working with equations where $a = 1$ in these tasks so that standard form for these tasks will be: $x^2 + bx + c = 0$.
- The following is called *factored form*: $(x - r_1)(x - r_2) = 0$. Note that they should not be told that r_1 and r_2 are solutions to the quadratic equation. Instead, they should discover this fact for themselves during the activities. If they ask why there are minus signs in the parentheses, they can be told that it doesn't matter if they are plus or minus but it is standard practice to write the general form using minus signs.
- They should know enough algebra, or review it as needed, to be able to multiply two binomials. For example, they should be able to multiply and simplify $(x + 3)(x - 5)$. Note that there is a task specifically designed to work with this.
- They should be informed that in these tasks they will only be solving quadratic equations that have integer coefficients and integer solutions.

A1.2.4 The Output Should Include Calculation Details

When a task asks a student to show their calculations it is important that the results are not hard coded, but are actually calculated in the program. For example, assume a student wants to show the calculations to verify that $x = 3$ is a solution to $x^2 - 9 = 0$. The following are two examples of how this may be done, one to avoid, and one to encourage:

Avoid this!
`print(f"{num}^2 - 9 = 0")`
Encourage this!
`print(f"{num}^2 - 9 = {num*num-9}")`

In the first example, the student sets the expression equal to 0, which assumes the solution is correct in the first place. In the second example, the expression is actually evaluated, so that errors can more easily be discovered by examining the output.

A1.2.5 Task Overview

The following is a brief overview of the tasks:

1. Write a program that asks the user to input an integer. The program should then output all possible signed factor pairs for that number.
2. Modify your previous program to now also output the sum of each factor pair. The calculation for the sum should be shown in each case. For example:
   `Number: 6 Factors: (1,6)  Sum: 1 + 6 = 7`
3. Add a function to your program that generates all integer combinations for two integers r_1 and r_2, where each integer can range from -3 to 3. For example: (-3,-3) (-3,-2) … (3,3). For each integer pair, output the pair and then the following two binomials as a product:
   $(x - r_1)(x - r_2)$. For example: $(-2,1)$: $(x + 2)(x - 1)$
4. Modify your previous function that prints out binomial pairs to include the simplified product of the two binomials as well. For example: $(3, -5)$: $(x - 3)(x + 5) = x^2 + 2x - 15$
5. Write a program that evaluates the expression $(x - 3)(x + 5)$ for all integer values of x from $x = -10$ to $x = 10$. Make note of any patterns you see in the output as well as when the expression evaluates to 0. The output should include the complete calculation when evaluating the expression.
6. Modify your function from task 3 to now turn the binomials into a quadratic equation in factored form by setting the product equal to 0. For example, instead of outputting $(x - 3)(x + 2)$ as it did in task 3, the function should now output $(x - 3)(x + 2) = 0$.
   The program should then print all the solutions to that equation, as well the calculations to verify that the solutions really are solutions to the equation.

7. Write a program that asks the user to input two integer *solutions* to a quadratic equation. The solutions can be positive or negative. The program should then output a quadratic equation in factored form that has those two solutions. The program should then output the same equation in standard form
8. Consider the following quadratic equation in standard form: $x^2 + bx + c = 0$. Write a program that outputs that equation with all integer combinations of b and c between -5 and 5. For example: $x^2 - 5x - 5 = 0, x^2 - 5x - 4 = 0, \ldots, x^2 + 5x + 5 = 0$ The program should then try and factor the associated quadratic equation. If the equation is factorable, the program should output the equation in standard form and factored form. For example: $x^2 - 5x + 4 = (x - 1)(x - 4) = 0$
9. Modify your previous program to also print out the solutions to the equations which have integer solutions. The program should show the calculation verifying the solutions using the factored form. For example:
   $x^2 - 5x + 4 = (x - 1)(x - 4) = 0$
   `Sol: x = 1 or x = 4`
   `Check: (1-1)(1-4)=0*-3=0 and (4-1)(4-4)=3*0=0`

A1.2.6 Task 1: Find All Signed Factor Pairs of an Integer

Problems addressed	Not understanding the connection between standard form and factored form. Incomplete understanding of factoring an integer.
Goal	This task lays the groundwork for the connection between factoring the constant term and factoring the equation. This task also aims at working towards consolidating knowledge elements related to factoring an integer.
Procepts	Both the concept of the factors of an integer as well as how to work with them numerically.
Programming benefits	Writing the program should help students deepen their understanding of factoring an integer.

Predict

Find all signed factor pairs of 6. For example: $2 \cdot 3 = 6$ and $-1 \cdot -6 = 6$. Is there a systematic way to find all the factor pairs? What would be different if the number was -6 instead of 6?

Code

Write a program that asks the user to input an integer. The program should then output all possible signed factor pairs for that number.

Example output

```
Enter an integer: 6
[[1, 6], [-1, -6], [2, 3], [-2, -3]]
```

Example Python program

```
def factor(num):
    factor_pairs = []
    for d in range(1,int(abs(num)**0.5)+1):
        if num % d == 0:
            factor_pairs.append([d,int(num/d)])
            factor_pairs.append([-d,-int(num/d)])
    return factor_pairs

fp = factor(int(input("Enter an integer: ")))
print(fp)
```

Notes on task 1

Important considerations and what to look out for

- Make sure the students include positive and negative factors.
- Make sure the program can handle factoring negative integers as well. In particular, look out for the loop bounds assuming the integer is positive.

- The program should be able to handle being asked to factor 0 in a reasonable way (this will later correspond to equations with no constant term).

Checking for divisibility

The easiest way to check for divisibility in many text-based languages is with the modulo operator **%**. The teacher can decide if they want students to think about the problem themselves a bit or just explain the use of the operator to them directly as this information will not affect the intervention. Note, however, that the students should first come up with the idea themselves of checking for divisibility.

Casting division to an int

Note that in some programming languages the results of division are automatically cast to floats, even when the result is an integer. To avoid problems when the expected result is an integer, it is advisable to cast the result to an integer. For example: `int(num/d)` or `num//d` instead of just `num/d`.

Skeleton code for task 1

If necessary, an outline of the function for this task can be provided to the students without compromising the important mathematics. This outline can include input parameters to the function, advice on storing the results, and the return statement returning the results. For example:

```
def factor(num):
    # this function is called with an integer to factor "num"
    # add code to find all factor pairs of this integer
    # found pairs can be stored in the following list
    factor_pairs = []
    return factor_pairs
```

A1.2.7 Task 2: Output the Sum of the Factor Pairs

Problems addressed	Not understanding the connection between standard form and factored form.
Goal	This task lays the groundwork for the connection between factoring the constant term and factoring the equation.

Code

Modify your previous program to now also output the sum of each factor pair. The calculation for the sum should be shown in each case. For example: `Number: 6 Factors: (1,6) Sum: 1 + 6 = 7`

Example output

```
Enter an integer: 6
Factors: (1,6)    Sum: 1 + 6 = 7
Factors: (-1,-6)  Sum: -1 + -6= -7
Factors: (2,3)    Sum: 2 + 3 = 5
Factors: (-2,-3)  Sum: -2 + -3 = -5
```

Example Python program

```
def print_factors_sums(factor_pairs):
    for fp in factor_pairs:
        f1 = fp[0]
        f2 = fp[1]
        s = f1 + f2
        print(f"Factors: ({f1},{f2})  Sum: {f1} + {f2} = {s}")
```

Notes on task 2

A potential issue with this task is how to print out the addition of negative numbers (see the example output above). Since later tasks will only use the sum as an internal calculation, there is no real need to spend any time on this issue. It is likely good enough to just include a space after the plus sign in the output for clarity

However, since the issue of adding negative problems will be a bigger issue in later tasks, it may be still be worth a quick discussion on the topic to start the students thinking about possible solution.

Skeleton code for task 2

Here is some code to align columns in Python:

```
fstr = f"Factors: ({f1},{f2})"
sstr = f"Sum: {f1}+{f2}={s}"
print(f"{fstr : <20} {sstr}")
```

A1.2.8 Task 3: Output Two Binomials

Goal	This task is designed to give the students a chance to deal with some technical issues involved in printing out binomials having to do with negative numbers. It also lays the groundwork for getting them used to seeing and working with factored form. Having this as its own task will help the students separate technical issues related to programming and programming related to mathematics.

Code

Add a function to your program that generates all integer combinations for two integers r_1 and r_2, where each integer can range from -3 to 3. For example: (-3,-3) (-3,-2) … (-2,1) … (3,3)

For each integer pair, output the pair and then the following two binomials as a product: $(x - r_1)(x - r_2)$

For example: (−2,1): $(x + 2)(x - 1)$

Example output

```
(-2,-1): (x + 2)(x + 1)
(-2,0): (x + 2)(x)
(-2,1): (x + 2)(x - 1)
```

Example additional Python code

```
def get_nm(num):
    if num < 0:
        return f" + {-num}"
    elif num == 0:
        return ""
    else:
        return f" - {num}"

def print_binomials():
    for r1 in range(-3,4):
        for r2 in range(r1,4):
            print(f"({r1},{r2}): (x{get_nm(r1)})(x{get_nm(r2)})")
```

Notes on task 3

This task is mainly about solving some technical issues that may arise when printing algebra expressions, but also about drawing awareness to special cases such as when one or both of the integers are zero or negative numbers.

Special cases

Students may initially ignore the special cases, which would result in output such as $(x - -3)$ and $(x - 0)$. It may be beneficial for the students to identify these "problems" and attempt to come up with solutions for them. The `get_nm()` function in the example above is one way of solving the problem. A solution such as this may be given to the

students by the teacher, but only after the students have become aware of the problem and made some attempt to solve it themselves.

The reason for allowing the students to first attempt a solution is to draw their attention to the fact that $(x + 3) = (x - -3)$ and that $x = (x - 0)$. Note that it is more important that they understand that $(x + 3) = (x - -3)$ than that $(x - -3) = (x + 3)$ because of the connection between the solutions and factored form.

Nested loops

The nested for-loops may also be difficult for the students and this solution can just be given to them directly if necessary. In the example above, the inner loop starts at the current location in the outer loop in order to avoid duplicates. However, for this task, duplicates don't matter. Students may find a similar solution to eliminate them on their own, if motivated, but there is no need to bring this to their attention if they don't see it themselves.

Note that some languages have specialized libraries and functions for creating combinations, such as `itertools` in Python, though it not clear that these would be any easier than a nested for-loop.

Print to the screen or to a file

Some programming environments limit how far back a user can scroll through output to the console. Since this program generates many output lines, and other tasks will generate even more, it may be a good time to introduce outputting to a file instead of to the console. If the students don't know how to do this themselves, the teacher can give them a solution.

The following is a simple way to do this in Python by creating a new print statement that outputs to a file automatically. The students can then use `fprint` instead of print in their program.

```
open("math.txt","w").close()
def fprint(pstr):
    print(pstr,file=open("math.txt","a"))
```

A1.2.9 Task 4: Output the Product of Binomials

Problems addressed	Lack of fluency with algebra necessary to connect standard form and factored form. Not understanding the connection between factored form and the coefficients in standard form.
Goal	This task is designed to allow the students to discover the relationship between factored form and the coefficients in standard form. This lays important groundwork for factoring quadratic equations as well as for understanding Vieta's formulas.
Procepts	Both the object of a factored quadratic equation as well as how to manipulate it algebraically. The connection between factored form and the coefficients in standard form.
Programming benefits	Writing the program should help students make explicit the connection between factored form and the coefficients in standard form. In order to write a program that converts factored form to standard form, they will need to write general rules for the connection between the binomials and the coefficients.

Predict

Multiply and simplify $(x-3)(x+5)$

Code

Modify your previous function that prints out binomial pairs to include the simplified product of the two binomials as well. For example: $(3,-5)$: $(x-3)(x+5) = x^2+2x-15$

Example output

```
(-2,0): (x + 2)(x) = x² + 2x
(-2,1): (x + 2)(x - 1) = x² + x - 2
(-2,2): (x + 2)(x - 2) = x² - 4
```

Example Python program

```
def print_binomials():
    for r1 in range(-3,4):
        for r2 in range(r1,4):
            bi = f"({r1},{r2}): (x{get_nm(r1)})(x{get_nm(r2)})"
            cofx = -(r1 + r2)
            con = r1*r2
            standard = f"x^2 + {cofx}x + {con}"
            fprint(f"{bi} = {standard}")
```

Notes on task 4

It is import for the teacher to visually inspect a number of different cases to make sure the standard form actually matches the given factored form. Students could check a few by hand themselves first to

get in the habit of checking their own results, but these should be confirmed by the teacher.

Sign issues with the coefficients

If the students are using the standardized minus sign for factored form, then they will need to be careful about the sign of the coefficient of the linear term in standard form since $b = -(r_1 + r_2)$.

Special cases

Cases where $b = 0$ or $c = 0$ in standard form should be handled in some way. Letting the students just output $x^2 + 2x + 0$ and $x^2 + 0x - 4$ instead of $x^2 + 2x$ and $x^2 - 4$ may obscure the value of considering these potentially simpler cases separately. If the students aren't handling these cases, their attention should be drawn to them so they can update their program to output them correctly.

Printing issues

There are a number of formatting issues to consider when printing in standard form. For the constant term, the problem is similar to that already discussed for printing the binomials.

There are a several special cases for the coefficient of the linear term. If the coefficient is 0, the linear term should not be printed at all instead of just printing $0x$. The other case to consider is when the coefficient is ± 1 in which case the 1 should not be printed, only $\pm x$.

The last issue is with the exponent for the quadratic term. Standard in-text notation would be to print it as `x^2`, however the teacher could introduce the Unicode notation `x\u00B2` which outputs x^2 which is easier to read.

Vieta's formulas

A discussion should be had with the students about their code and how it found the coefficient of the linear term and the constant term from factored form. This relationship should be stated explicitly. The students should be able to explain that the coefficient of the linear term is $-(r_1 + r_2)$ and the constant term is $r_1 r_2$.

A1.2.10 Task 5: Evaluating an Expression

Problems addressed	Difficulties evaluating expression. Difficulties checking the solutions to quadratic equations. Not understanding that both x's in factored form represent the same number.
Goal	This task is designed to help the students to discover the relationship between the solutions to a quadratic equation and the equation in factored form.
Procepts	Both the object of a factored quadratic equation as well as how to manipulate it algebraically. The connection between factored form and the solutions to the equation.
Programming benefits	Allowing the computer to take care of repeated calculations will allow the students to focus on understand the results of the calculations. Programming the calculations themselves will help students make the connection between the solutions and factored form.

Predict

Evaluate the expression $(x-3)(x+5)$ for $x=-2$ and then for $x=3$.

Code

Write a program that evaluates the expression $(x-3)(x+5)$ for all integer values of x from $x=-10$ to $x=10$. Make note of any patterns you see in the output as well as when the expression evaluates to 0. The output should include the complete calculation when evaluating the expression.

Example output

```
x = -6: (-6-3)(-6+5) = (-9)(-1) = 9
x = -5: (-5-3)(-5+5) = (-8)(0) = 0
x = -4: (-4-3)(-4+5) = (-7)(1) = -7
```

Example Python program

```
def evaluate():
    for x in range(-10,11):

        fprint(f" x = {x}: ({x}-3)({x}+5) = ({x-3})({x+5}) = {(x-3)*(x+5)}")
```

Notes on task 5

It is important that all calculation steps are shown in the output as in the example above.

There are several important aspects of the output that the students should identify in discussion:

- The expression is equal to zero exactly twice.
- The value of the expression appears to consistently move away from 0 as x moves away from the two roots. The teacher may opt to have the students graph the result here as well.
- The students should recognize the connection between +5 and -3 in the expression and the reason why the expression is equal to 0 for $x = -5$ and $x = 3$.

A1.2.11 Task 6: Find the Solutions to an Equation in Factored Form

Problems addressed	Misunderstanding the zero product property. Not understanding the connection between the solutions and the equation in factored form. Not understanding that both x's in factored form represent the same number.
Goal	This task is builds on the previous task to help the students discover the relationship between the solutions to a quadratic equation and the equation in factored form.
Procepts	Both the object of a factored quadratic equation as well as how to manipulate it algebraically. The connection between factored form and the solutions to the equation.
Programming benefits	In order to write this program, the students must understand the connection between the solutions and a quadratic equation in factored form.

Code

Modify your function from task 3 to now turn the binomials into a quadratic equation in factored form by setting the product equal to 0. For example, instead of printing $(x-3)(x+2)$ as it did in task 3, the function should now print $(x-3)(x+2)=0$.

The program should then print all the solutions to that equation, as well the calculations to verify that the solutions really are solutions to the equation.

Example output

```
(-3,1): (x + 3)(x - 1) = 0
Solutions: x = -3 or x = 1
Verify x = -3: (-3 + 3)(-3 - 1) = (0)(-4) = 0
Verify x = 1: (1 + 3)(1 - 1) = (4)(0) = 0
```

Example Python program

```
def rand():
    for r1 in range(-3,4):
        for r2 in range(r1,4):
            fprint(f"({r1},{r2:}): (x{get_nm(r1)})(x{get_nm(r2)}) = 0")
            fprint(f"Solutions: x = {r1} or x = {r2}")
            fprint(f"Verify             x             =             {r1}:
({r1}{get_nm(r1)})({r1}{get_nm(r2)}) =
                    ({r1-r1})({r1-r2}) = {(r1-r1)*(r1-r2)}")
            fprint(f"Verify             x             =             {r2}:
({r2}{get_nm(r1)})({r2}{get_nm(r2)}) =
                    ({r2-r1})({r2-r2}) = {(r2-r1)*(r2-r2)}\n")
```

Notes on task 6

It is important not only that the calculations are shown in the output, but that the calculations are actually done in the code as well.

Additional factors to look out for:

- Make sure the students explicitly identify the solutions to the equations in their output.
- Make sure the solutions are checked one at a time and not both at the same time.

A1.2.12 Task 7: Produce a Quadratic Equation From Its Solutions

Problems addressed	Misunderstanding the zero product property. Not understanding the connection between the solutions and the equation in factored form.
Goal	This task is designed to allow the students to discover the relationship between the solutions to a quadratic equation and the coefficients when in standard form.
Procepts	Both the object of a factored quadratic equation as well as how to manipulate it algebraically. Both the concept of the connection between the solutions to a quadratic equation and its factored form as well as how to use that connection.
Programming benefits	Writing the program should help students make explicit the connection the between the solutions to a quadratic equation and the equation in factored form as well as the relationship between the solutions and the coefficients in standard form.

Predict

The solutions to a quadratic equation are $x = 3$ and $x = 4$. How would you go about writing a quadratic equation in factored form that has those two solutions? How would you then write the same equation in standard form? Can you use the solutions to help you with both forms?

Code

Write a program that asks the user to input two integer solutions to a quadratic equation. The solutions can be positive or negative. The program should then output a quadratic equation in factored form that has those two solutions. The program should then output the same equation in standard form.

Example output

```
>>>
Enter two integer solutions to a quadratic equation.
Solution 1 = 3
Solution 2 = 4
(x - 3)(x - 4) = 0
x² - 7x + 12 = 0
```

Example Python program

```
print("Enter two integer solutions to a quadratic equation.")
sol1 = int(input("Solution 1 = "))
sol2 = int (input("Solution 2 = "))
print(f"(x - ({sol1}))(x - ({sol2})) = 0")
b = -(sol1+sol2)
c = sol1 * sol2
print(f"x\u00B2 + ({b})x + ({c}) = 0")
```

Notes on task 7

The students should be able to reuse the code they wrote in previous tasks to print quadratic equations in both factored and standard form, as well as converting from factored form to standard form. Special

issues with coefficients, including problems with signs, as well as coefficients of 1 and 0, should already have been dealt with in previous tasks. The main step here will be going from the solutions to factored form, in particular being careful about negative solutions.

A1.2.13 Task 8: Factor a Quadratic Equation With Integer Solutions

Problems addressed	Not understanding the connection between the equation in standard form and in factored form. Not understanding the connection between the solutions to a quadratic equation and the equation in factored form and standard form.
Goal	This task builds on the previous two tasks and allows students to apply the understanding gained there to actually factor a quadratic equation when possible.
Procepts	Both the concept of the connection between the solutions to a quadratic equation and its standard form as well as how to work with that connection numerically.
Programming benefits	Writing the program should help students deepen their understanding of the connection between the solutions to a quadratic equation and the coefficients of the equation in standard form. In particular, in order to create an algorithm that can factor an equation over the integers, the students need to have a deeper understanding of why the algorithm works.

Predict

How can you use the results of the previous activities to help you factor $x^2 + 5x + 6 = 0$? You may find it helpful to use the function from activities 1 and 2. What about $x^2 + 2x + 3$?

Code

Consider the following quadratic equation in standard form: $x^2 + bx + c = 0$

Write a program generates all integer combinations of b and c between -5 and 5. For example:

$x^2 - 5x - 5 = 0$

$x^2 - 5x - 4 = 0$

…

$x^2 + x + 2 = 0$

…

$x^2 + 5x + 5 = 0$

The program should then try and factor the associated quadratic equation. If the equation if factorable, the program should output the equation in standard form and factored form. For example:

$x^2 - 5x + 4 = (x - 1)(x - 4) = 0$

Example output

```
>>>
x² - 5x + 4 = (x - 1)(x - 4) = 0
x² - 4x - 5 = (x + 1)(x - 5) = 0
x² - 4x + 3 = (x - 1)(x - 3) = 0
x² - 4x + 4 = (x - 2)(x - 2) = 0
x² - 3x - 4 = (x + 1)(x - 4) = 0
x² - 3x + 2 = (x - 1)(x - 2) = 0
x² - 2x - 3 = (x + 1)(x - 3) = 0
x² - 2x + 1 = (x - 1)(x - 1) = 0
x² - 1x - 2 = (x + 1)(x - 2) = 0
x² - 4 = (x + 2)(x - 2) = 0
```

Example Python program

```
for b in range(-5,6):
    for c in range(-5,6):
        for div1 in range(1, int(abs(c)**0.5)+1):
            if c % div1 == 0:
                div2=c//div1
                if div1+div2==b:
                    nicePrint(b,c,div1,div2)
                if div1+div2==-b and b!=0:
                    nicePrint(b,c,-div1,-div2)
```

Notes on task 8

Nested loops

If nested loops are too hard for the students, they can be given the loops as skeleton code and just fill in the inner loop. It may be necessary to clarify the instructions as well.

Printing issues

As in previous tasks, there are potential printing issues here when printing in standard form with negative coefficients as well as with coefficients of 1 and 0. Note that only factorable equations should be printed.

Connecting to task 2

The students should be encouraged to use the results of task 2 as part of their program.

Special cases

Note that if the students are using task 2 (or the ideas learned there) to decide if an equation is factorable, they may run into problems when $c = 0$. The program should try and find all factor pairs of c, and then find the ones that sum to $-b$, but this will fail when $c = 0$. Depending on how it fails, the problem may not be noticeable to the students with the only result being these equations are not included in the final list. If the teacher sees that this problem is occurring, it is okay to point it out to the students and encourage them to handle these cases in a different manner. This should not pose too much of a problem since they can all be factored in the same way (excluding the case $x^2 = 0$): $x^2 + bx = x(x + b)$ with solutions $x = 0$ and $x = -b$.

Checking results

It will be helpful for the teacher to have a comprehensive list of the results to check against the student output to make sure nothing is missed. Depending on the programming environment, it may be easier to have the students output the results to a file since the console may hide some of the output if the list is too long. The teacher can just provide the students with code for writing to a file if necessary.

Skeleton code for task 8

Here is a code outline that can be given to the students for the nested loops

```
for b in range(-5,6):
    for c in range(-5,6):
        #check if x^2+bx+c can be factored
        #if no, do nothing
        #if yes, factor it and print in both forms
```

Complete list of factorable equations

Here is a complete list of the 31 equations which are factorable in the given range:

$x^2 - 5x = x(x - 5) = 0$
$x^2 - 5x + 4 = (x - 1)(x - 4) = 0$
$x^2 - 4x - 5 = (x + 1)(x - 5) = 0$
$x^2 - 4x = x(x - 4) = 0$
$x^2 - 4x + 3 = (x - 1)(x - 3) = 0$
$x^2 - 4x + 4 = (x - 2)(x - 2) = 0$
$x^2 - 3x - 4 = (x + 1)(x - 4) = 0$
$x^2 - 3x = x(x - 3) = 0$
$x^2 - 3x + 2 = (x - 1)(x - 2) = 0$
$x^2 - 2x - 3 = (x + 1)(x - 3) = 0$
$x^2 - 2x = x(x - 2) = 0$
$x^2 - 2x + 1 = (x - 1)(x - 1) = 0$
$x^2 - x - 2 = (x + 1)(x - 2) = 0$
$x^2 - x = x(x - 1) = 0$
$x^2 - 4 = (x + 2)(x - 2) = 0$
$x^2 - 1 = (x + 1)(x - 1) = 0$
$x^2 = x(x) = 0$
$x^2 + x - 2 = (x - 1)(x + 2) = 0$
$x^2 + x = x(x + 1) = 0$
$x^2 + 2x - 3 = (x - 1)(x + 3) = 0$
$x^2 + 2x = x(x + 2) = 0$
$x^2 + 2x + 1 = (x + 1)(x + 1) = 0$
$x^2 + 3x - 4 = (x - 1)(x + 4) = 0$
$x^2 + 3x = x(x + 3) = 0$
$x^2 + 3x + 2 = (x + 1)(x + 2) = 0$
$x^2 + 4x - 5 = (x - 1)(x + 5) = 0$
$x^2 + 4x = x(x + 4) = 0$
$x^2 + 4x + 3 = (x + 1)(x + 3) = 0$
$x^2 + 4x + 4 = (x + 2)(x + 2) = 0$
$x^2 + 5x = x(x + 5) = 0$
$x^2 + 5x + 4 = (x + 1)(x + 4) = 0$

A1.2.14 Task 9: Solve a Quadratic Equation With Integer Solutions

Problems addressed	Not understanding the connection between the equation in standard form and in factored form. Not understanding the connection between the solutions to a quadratic equation and the equation in factored form and standard form.
Goal	This task combines the understanding gained in all the previous tasks to solve a quadratic equation with integer solutions by factoring and using the zero product property.
Procepts	Both the concept of the connection between the solutions to a quadratic equation and its standard form as well as how to work with that connection numerically.
What does programming add?	Writing the program should help students deepen their understanding of the connection between the solutions to a quadratic equation and the coefficients of the equation in standard form. In particular, in order to create an algorithm that can factor an equation over the integers, the students will need to have a deeper understanding of why the algorithm works. Implementing a method for finding both solutions from factored form should help deepen this connection as well.

Predict

How can you use the results of the previous activity to help you solve $x^2 - 7x + 6 = 0$?

Code

Modify your previous program to also print out the solutions to the equations which have integer solutions. The program should show the calculation verifying the solutions using the factored form. For example:

```
x² - 5x + 4 = (x - 1)(x - 4) = 0
Sol: x = 1 or x = 4
Check: (1-1)(1-4) = 0 * -3 = 0 and (4-1)(4-4) = 3 * 0 = 0
```

Example output

```
>>>
x² - 5x + 4 = (x - 1)(x - 4) = 0  Sol: x = 1 or x = 4
Check: 0 * -3 = 0 and 3 * 0 = 0
x² - 4x + 3 = (x - 1)(x - 3) = 0  Sol: x = 1 or x = 3
Check: 0 * -2 = 0 and 2 * 0 = 0
x² - 4x + 4 = (x - 2)(x - 2) = 0  Sol: x = 2 or x = 2
Check: 0 * 0 = 0 and 0 * 0 = 0
x² - 3x - 4 = (x + 1)(x - 4) = 0  Sol: x = -1 or x = 4
Check: 0 * -5 = 0 and 5 * 0 = 0
```

Example Python program

```
for b in range(-5,6):
    for c in range(-5,6):
        for div1 in range(1, int(abs(c)**0.5)+1):
            if c % div1 == 0:
                div2=c//div1
                if div1+div2==b:
                    nicePrint(b,c,div1,div2)
                if div1+div2==-b and b!=0:
                    nicePrint(b,c,-div1,-div2)
```

A1.3 Tasks Used Intervention 3

Unlike with the previous two interventions, there was not time to create a comprehensive task guide before the third intervention. Instead, the limited time was used to make changes to the task texts that would be given to the students as well as to create example programs and functions. These changes were heavily based on practical problems encountered during the second intervention and mostly involved ways of trying to ameliorate the anticipated problems that would occur due to lack of programming experience among the students. This was an attempt to better apply the following task design criteria (see section 9.3):

- Students need to be encouraged to actively reflect on the results of the activities as they will not do so spontaneously.
    - If the programming requirements were too difficult, the students would have a harder time reflect on the results.
- Group work is important as it fosters discussion and debate.
    - If the programming requirements were too difficult, then group discussions would be harder and there would be more of a tendency for the one who knows the most programming to “take over”.
- Encourage cognitive conflict.
    - Cognitive conflict can only be useful where the students understand enough to have some kind of expectation about outcome.
- Encourage curiosity.
    - Curiosity is also dependent on the students feeling comfortable enough with the task.
- Don't overcomplicate tasks.
    - If the programming elements are too difficult then the task is overcomplicated.

During the intervention, some of the tasks were changed and new example programs and functions were created on the fly as I became increasingly aware of how difficult the programming aspects were for the students.

A1.3.1 Task Overview

The follow is a brief overview of the programming preparation tasks:

1. Print out all integers between -3 and 3.
2. Print out "$(x - a)$" for all values of a between -3 and 3.
3. Print out all combinations of two integers between -3 and 3.
4. Print out all "$(x - a)(x - b) = 0$" for all combinations of integers a and b between -3 and 3.
5. Print out "$x\text{^}2 + px + q = 0$" for all combinations of integers p and q between -3 and 3.

The following is a brief overview of the main tasks:

1. Write a program which asks the user for an integer, positive or negative. The program should then print out all of the integer's factor pairs (both positive and negative).
2. Change the program from task 1 so that it also prints out the sum of every factor pair. The calculation for each sum should also be shown.
3. Write a program which evaluates the following expression for all integer values of x from $x = -7$ to $x = 7$: $(x - 3)(x + 5)$.
4. Change the program from programming preparation task 4 so that the program also prints out all solutions to the equation.
5. Change the program from task 4 so that the program multiplies and simplifies the two binomials and prints out the resulting expression.
6. Write a program which asks the user for two integers, positive or negative, which will be solutions to a quadratic equation. The program should then print out a quadratic equation in factored form which has the given integers as solutions. The program should then print out the same equation in standard form.
7. Change the program from programming preparation task 5 so that p and q take on all integer value between -5 and 5 instead of between -3 and 3. The program should then try and factor every equation. If it can be factored, the program should print out the equation in both factored form and in standard form. If it can't be factored the program should not print it out at all.

8. Change the program from task 7 so that it also print out the solutions to the equations.

A1.3.2 Programming Preparation Tasks

The first five tasks were intended to help the students get comfortable with some basic programming ideas while creating programs that would be useful during later mathematical tasks. These tasks had very little mathematical content to allow the students to focus on the technical aspects of programming. Example programs for several of these tasks were created in advance and the plan was to give them to the students after they had struggled a bit on their own with them. The hope was that separating these aspects would allow them to focus more on the mathematics in the later tasks.

Note that programming preparation task 5 was never shown to the students due to how much they struggled with the previous tasks.

Programming preparation task 1

Print out all integers between -3 and 3. Example output:

```
-3, -2, -1, 0, 1, 2, 3
```

Programming preparation task 2

Print out "$(x - a)$" for all values of a between -3 and 3. Include the value of a in every row. Example output:

```
a = -3: (x + 3)
a = -2: (x + 2)
...
```

Programming preparation task 3

Print out all combinations of two integers between -3 and 3. Example output:

```
(-3,-3)
(-3,-2)
...
```

Programming preparation task 4

Print out all "$(x - a)(x - b) = 0$" for all combinations of integers a and b between -3 and 3. Include the values of a and b in every row. Example output:

```
(-3,-3): (x + 3)(x + 3) = 0
(-3,-2): (x + 3)(x + 2) = 0
...
```

Programming preparation task 5

Print out "$x^2 + px + q = 0$" for all combinations of integers p and q between -3 and 3. Include the values of p and q in every row. Example output:

```
(-3,-3): x² - 3x - 3 = 0
(-3,-2): x² - 3x - 2 = 0
...
```

A1.3.3 The Main Tasks

The following were the rest of the tasks intended to be given to the students (they did not finish all of them in time) which had more of a mathematical focus than those listed above. The different parts of the tasks have been renamed to *With Pen and Paper* and *On the computer* to better reflect how they were presented to the students. Example programs are not provided here since they do not differ from those provided in the previous interventions.

A1.3.4 Task 1: Factoring an Integer

Problems addressed	Not understanding the connection between standard form and factored form. Incomplete understanding of factoring an integer.
Goal	This task lays the groundwork for the connection between factoring the constant term and factoring the equation. This task also aims at working towards consolidating knowledge elements related to factoring an integer.
Procepts	Both the concept of the factors of an integer as well as how to work with them numerically.
Programming benefits	Writing the program should help students deepen their understanding of factoring an integer.

With Pen and Paper
Find all factor pairs of the number 6. Include both positive and negative factors. For example: $2 \cdot 3 = 6$ and $-1 \cdot -6 = 6$. What would be different if the number was -6 instead of 6? Is there a systematic of finding all factor pairs?
On the Computer
Write a program which asks the user for an integer, positive or negative. The program should then print out all of the integer's factor pairs (both positive and negative).
Example output
`Enter an integer: 6` `[[1, 6], [-1, -6], [2, 3], [-2, -3]]`

A1.3.5 Task 2: Find the Sum of the Factors

Problems addressed	Not understanding the connection between standard form and factored form. Incomplete understanding of factoring an integer.
Goal	This task lays the groundwork for the connection between factoring the constant term and factoring the equation. This task also aims at working towards consolidating knowledge elements related to factoring an integer.
Procepts	Both the concept of the factors of an integer as well as how to work with them numerically.
Programming benefits	Writing the program should help students deepen their understanding of factoring an integer.

On the Computer
Change the program from task 1 so that it also prints out the sum of every factor pair. The calculation for each sum should also be shown.
Example output
`Number: 6 Factors: (1,6) Sum: 1 + 6 = 7`

A1.3.6 Task 3: Evaluating a Quadratic Equation in Factored Form

Problems addressed	Not understanding the connection between factored form and the solutions to a quadratic equation. Not understanding that there are two solutions to a quadratic equation.
Goal	This task is intended to help the students see why there are two solutions to a quadratic equation as well as laying the groundwork for the connection between factored form and the solutions.
Procepts	An equation in factored form as an object and how to work with it numerically.
Programming benefits	This task focuses more on the results than the act of programming. The students take use the computer's ability to rapidly perform many calculations and display the results.

On the Computer
Write a program which evaluates the following expression for all integer values of x from $x = -7$ to $x = 7$: $(x - 3)(x + 5)$ The program should evaluate and then print out the value of the expression for each value of x. The program should also show the calculations. Do you notice any patterns in the results? When is the expression equal to 0? Sketch a graph of the values in a coordinate system (by hand) and compare the graph with your observations above.
Example output
`x=-7: (-7-3)(-7+5)=(-10)(-2)=20`

A1.3.7 Task 4: Solving a Quadratic Equation in Factored Form

Problems addressed	Not understanding the connection between factored form and the solutions to a quadratic equation. Not understanding that there are two solutions to a quadratic equation.
Goal	This task is intended to help the students deepen their understanding of the connection between the solutions to a quadratic equation and that equation in factored form.
Procepts	An equation in factored form as an object and how to work with it numerically.
Programming benefits	While programming the calculations to verify their solutions they should gain a deeper understanding of the connections between the solutions to a quadratic equation and that equation in factored form. This will also reinforce the importance of checking each solution independently.

With Pen and Paper

Can you find any values of x which would make the following equation true? $(x-2)(x+3)=0$

How many solutions are there?

Can you explain exactly how you have found them?

On the Computer

Change the program from programming preparation task 4 so that the program also prints out all solutions to the equation as well as the calculations verifying that the solutions actually work. It can be helpful here to reflect over results from earlier tasks.

Example output

```
(-2, 1): (x + 2)(x - 1) = 0
Solutions: x = -2 or x = 1
Verify:
x = -2: (-2 + 2)(-2 - 1) = (0)(-3) = 0
x = 1: (1 + 2)(1 - 1) = (3)(0) = 0
```

A1.3.8 Task 5: Changing From Factored Form to Standard Form

Problems addressed	Difficulties simplifying the product of two binomials. Not understanding the connection between factored form and standard form.
Goal	The students should gain a better understanding of what they do when combining the linear terms in the product of two binomials. They should also start to develop an understanding of the connection between factored form and the coefficients in standard form.
Procepts	Factored form and standard form as algebraic objects and as expression that can be algebraically manipulated and worked with numerically.
Programming benefits	The act of the writing the program should help them make explicit the process used to find the coefficient of the linear term when multiplying two binomials. It should also start to make explicit the connection between the coefficients in standard form and factored form as well as the solutions.

With Pen and Paper

Multiply and simplify: $(x-3)(x+5)$

Try and explain exactly what you did. Be as detailed and specific as you can.

On the Computer

Change the program from task 4 so that the program multiplies and simplifies the two binomials and prints out the resulting expression. You can use the code from programming preparation 5.

You don't need to change any other parts from task 4.

Example output

```
(-2, 1): (x + 2)(x - 1) = x2 + x - 2 = 0
Solutions: x = -2 or x = 1
Verify:
x = -2: (-2 + 2)(-2 - 1) = (0)(-3) = 0
x = 1: (1 + 2)(1 - 1) = (3)(0) = 0
```

A1.3.9 Task 6: Generating a Quadratic Equation from Two Solutions

Problems addressed	Difficulties simplifying the product of two binomials. Not understanding the connection between factored form and standard form.
Goal	The students should gain a better understanding of what they do when combining the linear terms in the product of two binomials. They should also start to develop an understanding of the connection between factored form and the coefficients in standard form.
Procepts	Factored form and standard form as algebraic objects and as expression that can be algebraically manipulated and worked with numerically.
Programming benefits	The act of the writing the program should help them make explicit the process used to find the coefficient of the linear term when multiplying two binomials. It should also start to make explicit the connection between the coefficients in standard form and factored form as well as the solutions.

With Pen and Paper

Write down a quadratic equation in factored form which has solutions $x = 3$ and $x = 4$. Then write the same equation in standard form.

How have you used the solutions to write the equation in standard form?

Do you see the connection between the solutions and the coefficients in standard form?

Write down a quadratic equation in factored form which has solutions $x = -3$ and $x = 1$. Then write the same equation in standard form.

On the Computer

Write a program which asks the user for two integers, positive or negative, which will be solutions to a quadratic equation. The program should then print out a quadratic equation in factored form which has the given integers as solutions. The program should then print out the same equation in standard form.

Example output

```
Enter an integer solution: 2
Enter another integer solution: -3
(x - 2)(x + 3) = x² + x - 6 = 0
```

A1.3.10 Task 7: Factoring a Quadratic Equation in Standard Form

Problems addressed	Not understanding the connection between the equation in standard form and in factored form. Not understanding the connection between the solutions to a quadratic equation and the equation in factored form and standard form.
Goal	This task builds on the previous two tasks and allows students to apply the understanding gained there to actually factor a quadratic equation when possible.
Procepts	Both the concept of the connection between the solutions to a quadratic equation and its standard form as well as how to work with that connection numerically.
Programming benefits	Writing the program should help students deepen their understanding of the connection between the solutions to a quadratic equation and the coefficients of the equation in standard form. In particular, in order to create an algorithm that can factor an equation over the integers, the students need to have a deeper understanding of why the algorithm works.

With Pen and Paper
Use the results from previous tasks to write $x^2 + 5x + 6 = 0$ in factored form. Can you do the same for $x^2 + 2x + 3 = 0$?
On the Computer
Change the program from programming preparation task 5 so that p and q take on all integer value between -5 and 5 instead of between -3 and 3. The program should then try and factor every equation (rewrite it in factored form). If it can be factored, the program should print out the equation in both factored form and in standard form. If it can't be factored the program should not print it out at all.
Example output
`x²-5x+4=(x-1)(x-4)=0`

A1.3.11 Task 8: Solving a Quadratic Equation in Standard Form

Problems addressed	Not understanding the connection between the equation in standard form and in factored form. Not understanding the connection between the solutions to a quadratic equation and the equation in factored form and standard form.
Goal	This task combines the understanding gained in all the previous tasks to solve a quadratic equation with integer solutions by factoring and using the zero product property.
Procepts	Both the concept of the connection between the solutions to a quadratic equation and its standard form as well as how to work with that connection numerically.
Programming benefits	Writing the program should help students deepen their understanding of the connection between the solutions to a quadratic equation and the coefficients of the equation in standard form. In particular, in order to create an algorithm that can factor an equation over the integers, the students will need to have a deeper understanding of why the algorithm works. Implementing a method for finding both solutions from factored form should help deepen this connection as well.

With Pen and Paper

Use the results and methods from previous tasks to solve the following equation: $x^2 - 7x + 6 = 0$

On the Computer

Change the program from task 7 so that it also prints out the solutions to the equations which could be factored. The program should also print out the calculations that verify that the solutions are correct

Example output

```
x^2- 3x- 4 = (x+ 1)(x - 4) = 0
Solutions: x = -1 or x = 4
Verify:
x=-1:  (-1+1)(-1-4) = 0 * -5= 0
x=4:  (4+1)(4-4) = 5 * 0 = 0
```

A1.3.12 Final Task on Paper: Applying the Lessons

Try and solve the following equations by factoring them without using the computer.

Equation 1: $x^2 - 5x - 24 = 0$

Equation 2: $x^2 + 5x + 12 = 0$

A1.3.13 Example Programs and Functions

The following example programs, functions and skeleton code were provided for the students in intervention 3 at various stages during the tasks. Some were created in advance, while others were created on the fly during the lessons when the need was realized.

A1.3.13.1 Code for Programming Preparation Task 3

It was quickly apparent that nested loops were too difficult for the student so they were given the following code to examine and try and understand.

```
for r1 in range(-3,4):
    for r2 in range(-3,4):
        print(f"({r1},{r2})")
```

A1.3.13.2 Functions for Printing Factored and Standard Form Nicely

After realizing how much difficulty the students were having with basic programming, I started giving them more and more sample code to modify. Two important functions included in most of these examples were `get_factoredform()` and `get_standardform()`, each of which took two parameters and returned a nicely formatted string that could be used in output.

Since the most important part of the relevant tasks would be determining which parameters to send to this functions, taking care of the formatting for them didn't seem likely to interfere with the point of the lessons.

A1.3.13.3 Code for Task 3 on the Computer

In this task it was the actual output of the program that was of interest, as opposed to the creation of the program itself, so it seemed a small step to provide them with the following code for this task. The function takes as input the values of x, a and b and then returns a string showing the calculation to evaluate the expression $(x - a)(x - b)$.

```
def get_binomprodukt(x,a,b):
    if a > 0:
        pstr = f"({x}-{a})"
    elif a < 0:
        pstr = f"({x}+{abs(a)})"
    else: #a=0 här
        pstr = f"({x})"
    if b > 0:
        pstr = pstr + f"({x}-{b})"
    elif b < 0:
        pstr = pstr + f"({x}+{abs(b)})"
    else: #b=0 här
        pstr = psrt + f"({x})"
    pstr = pstr + f" = ({x-a})({x-b}) = {(x-a)*(x-b)}"
    return pstr
```

A1.3.13.4 Code for Task 4 on the Computer

This program was created between the first and second lesson when it became apparent just how much trouble the students were actually having with technical aspects of programming. This program incorporates the function for creating a string to print factored form. The rest of the program is mostly complete, aside from the details of the final print statements. The strings to be printed are formatted, but the contents are filled with 1's as placeholders to show the students where they need to change the code. The task was then changed to figuring out what needed to be added to the code to the replace all the 1's and the text "Replace this text with the solutions".

```
for r1 in range(-3, 4):
    for r2 in range(-3, 4):
        ff = get_factorform(r1,r2)
        row1 = f"({r1}, {r2}): {ff} = 0"
        print(row1)

        #write code for finding the solutions here
        row2 = "Solutions: "
        row2 = row2 +"Replace this text with the solutions"
        print(row2)

        print("Verify:")

        #write code here to verify the solutions
        #I wrote in 1's as place holders for where calculations and
        #variables should be included.
        #Replace all 1's with the correct variables or calculations
        rad4 = f"x={1}: ({1}-{1})({1}-{1}) = ({1})({1}) = {1}"
        rad5 = f"x={1}: ({1}-{1})({1}-{1}) = ({1})({1}) = {1}"
        print(rad4)
        print(rad5)
        print()
```

A1.3.13.5 Code for Task 6 on the Computer

This program was created between the second and third lessons and incorporates the functions for creating a string to print in factored form and in standard form. The rest of the program is mostly complete aside from which parameters to pass to these to functions. The task was changed to deciding how to use the solutions entered by the user to determine what to pass to the functions. Question marks appear in the code where the students were asked to make changes. The final print statement did not need to be changed.

```
r1 = int(input("Enter an integer solution: "))
r2 = int(input("Enter another integer solution: "))

factor = get_factorform(?,?)
standard = get_standardform(?,?)

print(f"{factor} = {standard} = 0")
```

A1.3.13.6 Code for Task 7 on the Computer

This program was created between the second and third lessons and incorporates the functions for creating a string to print in factored form and in standard form. Task 7 originally asked the students to modify their code from programming preparation task 5, however, due to the fact that they were never given this task to work on, in conjunction with their previously acknowledged programming difficulties, they were presented with this code to modify instead.

This program also includes the code to find the factor pairs of a given integer already built into it. However, the students had to figure out what variable to use here as the integer to be factored, based on the given coefficients for the quadratic equation. Code to calculate the sum of the factors was also included in the code given to them, but the students had to then decide what to do with the factors and sum of factors and how to connect that to sending parameters to the `get_factorform` and `get_standarform` functions.

```
for p in range(-5,6):
    for q in range(-5,6):
        quad = get_standardform(p,q)
        #what should ? be here? This number will be factored
        num = ?
        for n in range(1,num+1):
            if num%n==0: #if num is divisible by n then n is a factor
                fac1 = n
                fac2 = num//n
                #negative factors now
                fac3 = -n
                fac4 = -num//n
                #calculate the sums of the factors
                sum12 = fac1 + fac2
                sum34 = fac3 + fac4
                #What do you need to check here?
                #The following commands print out the results
                #factorform = get_factorform(r1,r2)
                #print(f"{quad} = {factorform}) = 0"
```

Appendix 2 Coded Transcription Lines

This appendix contains the identified epistemic actions connected to the different constructs, including previously formed constructs. An analysis of the coded transcriptions, as well as a description of the epistemic actions, appears in the main text. The epistemic actions identified were recognizing (R), building-with (B), and constructing (C).

Note that the utterances from the students in Swedish have been translated to English. All translations are my own.

Some of the epistemic actions appeared to be driven by the need to write a program to complete the task. Rows containing these epistemic actions have been highlighted in the tables below.

A2.1 Coded Lines From Intervention 1

The following section list the epistemic actions identified in the first intervention.

A2.1.1 C_0: The Zero Product Property

The following utterances show the epistemic actions which lead to C_0. Recognizing and building with actions labeled R_0 and B_0 refer to previous versions of this construct.

Line	Name	Utterance	Action
		Task 1	
3	S1	You can say if *number* is just 2 and the other is 0 then the product is 0.	R_0
9	S1	Just use 0 and it's done.	B_0
10	S2	Nothing else will make it 0 between them.	B_0
13	S1	Not when it's multiplication.	B_0
14	S2	No, nothing else.	B_0
27	S1	I don't think it's possible to do it with anything other than zero to get a zero for the product.	B_0
31	S2	The product is 8. There's almost nothing else you can times with so it will be …	C_0

Task 2			
112	S1	Yeah, so … no matter what we should always make it equal to 0. Because …	C_0
113	S2	Yes, because it will always be 0.	R_0
Task 3			
334	S1	Okay. Ah, yeah. So for this one it should be negative 3 and negative 4.	R_0
Task 6			
665	S1	Okay. So negative 2, right?	B_0
666	S2	Yes.	B_0
667	S1	x can be equal to 3 … hmmm … can be equal to …	C_0
668	S2	Nothing more, right?	C_0
669	S1	What else can it be?	C_0
670	S2	I don't know.	C_0
679	S2	This is just equal to 0.	C_0
680	S1	Times … this … it's equal to 0. It doesn't matter, it is always equal to 0.	C_0

A2.1.2 C_e: Understanding Equations and Expressions

The following utterances show the epistemic actions which lead to C_e. Recognizing and building with actions labeled R_e and B_e refer to previous versions of this construct.

Line	Name	Utterance	Action
		Task 2	
116	S1	It's not an equation. It's a …	R_e
117	S2	No, right, it's not one.	R_e
118	S1	Or yeah, it is one.	R_e
121	S1	It's pretty clear that you can write … if we take … x … times minus two, for example. But that is like one way. It works. If we enter a positive number so that x is two then this works. Right?	B_e
122	S1	If x is positive then it works. If x is negative, then it doesn't work because this has to be …	B_e
131	S1	Yeah. It doesn't represent it because it isn't two. It has to be a two for [*mumbles*] can't change. Then it has to be a two. But … the problem here is if x is a negative number then it doesn't work. So it has to work in some way for both positive and negative.	C_e
149	S1	You can … I don't know … we can expand here if it's possible or not. But otherwise we can write … the same but with x plus two since it then works for negative. But then we have two which work …	C_e
		Task 3	
331	S1	So solutions are the roots … so …	R_e
332	S2	I think so.	R_e
365	S1	But I don't think so … but then … it would be more x's than it would be here.	C_e
368	S1	That's not right. But … I think … no. But … if we could start with something like this and then see how they got here then it would have been much easier.	C_e
369	S2	Yeah. I don't really understand the connection.	C_e
372	S1	But wait … I realized something I did wrong. But …	C_e
373	S1	… is two x.	C_e
374	S2	I don't really understand.	C_e

375	S1	Yeah, but … wait a little … it's … eh … we … x squared … outside … three x … inside two x … but it's still …	C_e
376	S2	… what does it get …	C_e
377	S1	The six should be like this … yeah, it's last … okay.	C_e
378	S2	Yeah.	C_e
379	S1	So it's just six there. And then it's this …	C_e
380	S1	And, yeah, that's correct. But then it's … from one here … did it completely wrong … it should be …	C_e
381	S1	… outside … minus … four x … don't know why I wrote that. It's been a long time since I did it. Inside … negative three … x … last … plus 12 … and then it's this x … minus … I think that … I think this is it. But I don't know. Maybe did it wrong.	C_e
382	S2	I don't know. This isn't right. But this is right. So if you just … FOIL … expand out then it becomes standard form since this is standard form.	C_e

A2.1.3 C_{2s}: Two Solutions to a Quadratic Equation

The following utterances show the epistemic actions which lead to C_{2s}.

Line	Name	Utterance	Action
		Task 1	
67	S2	There are two solutions every time.	R_e, B_e
68	S1	Yeah, there are.	R_e, B_e
		Task 2	
237	S1	x there can be two.	R_o, B_e
239	S2	It can't be negative?	B_o, B_e
240	S1	No … it also works if it's zero.	B_o, B_e
241	S2	If it's zero, yes. That's true.	B_o, B_e
255	S1	It's zero here and if x is negative three, this is negative two and then it's zero.	B_o, B_e
257	S2	Yeah.	B_o, B_e
258	S1	If x is negative one then it's zero and this is two and two times zero is still equal to zero.	B_o, B_e

Task 3			
334	S1	Okay. Ah, yeah. So for this one it should be negative 3 and negative 4.	B_o, B_e
335	S2	Yeah.	B_o, B_e
336	S1	So we should do the same thing but … so … if … x … if x is three here it's zero. So we can take … x … minus four … then this work with … equal to zero.	B_o, B_e, C_{2s}
337	S2	Yeah.	B_o, B_e, C_{2s}
Task 6			
665	S1	Okay. So negative 2, right?	B_o, B_e, C_{2s}
666	S2	Yes.	B_o, B_e, C_{2s}
667	S1	x can be equal to 3 … hmmm … can be equal to …	B_o, B_e, C_{2s}
668	S2	Nothing more, right?	B_o, B_e, C_{2s}
669	S1	What else can it be?	B_o, B_e, C_{2s}
670	S2	I don't know.	B_o, B_e, C_{2s}

693	S1	So it's the just the same as it already says, I think.
		`int solution_first_pair = - f.first_pair;`
		`int solution_second_pair = -f.second_pair;`
			C_{2s}

697	S1	We have to show our calculations … we can write equal to zero here so we actually solve something. `printf("(%d + %d)(%d %d)=0\n", solution_first_pair, f.first_pair, solution_first_pair, f.second_pair`	C_{2s}
699	S1	Now we've got the first answer. We have to do the same thing with the other one.	C_{2s}
700	S2	It's exactly the same there.	C_{2s}

A2.1.4 C_{mb}: Multiplying Two Binomials

The following utterances show the epistemic actions which lead to C_{mb}. Recognizing and building with actions labeled R_{mb} and B_{mb} refer to previous versions of this construct.

Line	Name	Utterance	Action
		Task 3	
340	S1	So, we get … you can … you probably just use FOIL. Possibly.	R_{mb}
341	S2	Yeah. First …	R_{mb}
348	S1	First … outside … times … but … okay … does it really work?	B_e, B_{mb}
349	S2	Yes, I think so.	B_e, B_{mb}
350	S1	… inside … no …	B_e, B_{mb}
351	S2	I don't think that's it …	B_e, B_{mb}
360	S1	But you just need to use the distributive property? Jag guess. Is that right?	B_e, B_{mb}
361	S2	Yes.	B_e, B_{mb}
365	S1	But I don't think so … but then … it would be more x's than it would be here.	C_{mb}
368	S1	That's not right. But … I think … no. But … if we could start with something like this and then see how they got here then it would have been much easier.	C_{mb}
369	S2	Yeah. I don't really understand the connection.	C_{mb}
372	S1	But wait … I realized something I did wrong. But …	C_{mb}
373	S1	… is two x.	C_{mb}
374	S2	I don't really understand.	C_{mb}
375	S1	Yeah, but … wait a little … it's … eh … we … x squared … outside … three x … inside two x … but it's still …	C_{mb}
376	S2	… what does it get …	C_{mb}
377	S1	The six should be like this … yeah, it's last … okay.	C_{mb}
378	S2	Yeah.	C_{mb}
379	S1	So it's just six there. And then it's this …	C_{mb}

380	S1	And, yeah, that's correct. But then it's … from one here … did it completely wrong … it should be …	C_{mb}
381	S1	… outside … minus … four x … don't know why I wrote that. It's been a long time since I did it. Inside … negative three … x … last … plus 12 … and then it's this x … minus … I think that … I think this is it. But I don't know. Maybe did it wrong.	C_{mb}
382	S2	I don't know. This isn't right. But this is right. So if you just … FOIL … expand out then it becomes standard form since this is standard form.	C_{mb}

A2.1.5 C_{fi}: Factors of an Integer

The following utterances show the epistemic actions which lead to C_{fi}. Recognizing and building with actions labeled R_{fi} and B_{fi} refer to previous versions of this construct.

Line	Name	Utterance	Action
		Task 4	
479	S1	You can divide 6 by different things. That's how we get this	R_{fi}
480	S2	Yeah.	R_{fi}
481	S1	For example, if we divide … it has to be something without a decimal …	B_{fi}
482	S2	Yeah.	B_{fi}
484	S1	No, it should be like this.	B_{fi}
485	S2	What's the fastest way? Dividing by 2 is easy, you get what you can times with 2. But it doesn't work for odd numbers.	C_{fi}

496	S1	Wait, it shouldn't be x there … if 2 times 3 … it's … wait, I have to think.
		`for (int i = 0; i < x; i++) {`
		`  if (x % i == 0) {`
		`  }`
		`}`
			C_{fi}

498	S1	Let's just test it. *Notes - get division by 0 error and realize can't start loop at 0.*	C_{fi}
502	S1	Oh, we can fix that later. We got it right there. It does the same thing but … both negative.	C_{fi}
503	S2	Exactly. There we have all of them.	C_{fi}

510	S1	That should also work if we write negative 6.	C_{fi}
511	S2	Yeah.	C_{fi}
514	S1	Then we have to it once to the right and once to the left.	C_{fi}
515	S2	Yeah.	C_{fi}
516	S1	Because they are actually different.	C_{fi}
517	S2	Yeah.	C_{fi}

519	S1	It should work with positive numbers. We can just test it to see if it works.
		`int x_div = x/i`
		`solutions.push_back(i, x_div, i+x_div)`
		`solutions.push_back(-i, -x_div, i+x_div)`
			C_{fi}

532	S1	So we make this positive and when we find them we for make one negative and then the other.	C_{fi}

A2.1.6 C_{ffs}: Connection Between Factored Form and Solutions

The following utterances show the epistemic actions which lead to C_{ffs}.

Line	Name	Utterance	Action
		Task 2	
250	S1	Same as there … wait. It's … it could be negative three, right?	R_o, R_e
251	S2	It's zero and then … yeah, it can be negative three there.	R_o, R_e
253	S1	Yeah, it could be negative three and it could be … negative one also, right?	B_o, B_e
255	S1	It's zero here and if x is negative three, this is negative two and then it's zero.	B_o, B_e
257	S2	Yeah.	B_o, B_e
258	S1	If x is negative one then it's zero and this is two and two times zero is still equal to zero.	B_o, B_e
		Task 3	
334	S1	Okay. Ah, yeah. So for this one it should be negative 3 and negative 4.	R_o, R_e
335	S2	Yeah.	R_o, R_e

336	S1	So we should do the same thing but … so … if … x … if x is three here it's zero. So we can take … x … minus four … then this work with … equal to zero.	B_o, B_e
337	S2	Yeah.	B_o, B_e
403	S2	It's easier to make a function that makes solutions in factored and then a function that makes solutions in standard.	R_p
406	S1	Let's do that. And we can use both functions later for other things if we want to.	B_p
407	S2	Yes, exactly.	B_p
409	S2	Factored is really easy, it's just changing …	B_p
410	S1	You should print it out … it should be fairly simple.	B_p
414	S2	This should be … if they enter something positive it should be negative …	C_{ffs}
415	S1	Yeah.	C_{ffs}
416	S2	And the other way around.	C_{ffs}
418	S2	If x is less than zero it should be … otherwise it should be … no! It will always be like that!	C_{ffs}
419	S1	Yeah.	C_{ffs}
420	S2	So it doesn't even need to be …	C_{ffs}
421	S!	No.	C_{ffs}
429	S1	It should also be plus here instead of negative.	C_{ffs}
431	S1	You can make it look nicer but it would just be … it's not necessary.	C_{ffs}
433	S1	Actually plus doesn't need to be here if it's minus.	C_{ffs}
434	S2	No.	C_{ffs}
Task 4			
523	S2	Now we get the right sum anyways.	B_{fi}
539	S1	It has to … times …	B_{fi}, C_{ffs}
540	S2	Yeah. I kind of think …	B_{fi}, C_{ffs}
541	S1	Aha. It should be added together to 13 but times to …	B_{fi}, C_{ffs}
542	S2	To 12.	B_{fi}, C_{ffs}

543	S1	Four, three. No. It should …	B_{fi}, C_{ffs}
551	S2	Yeah, it's 1 and 12.	B_{fi}, C_{ffs}
552	S1	Aha. I thought it should add to 12, but it should times to 12.	B_{fi}, C_{ffs}
553	S2	Yes, and then … times to 12 … and …	B_{fi}, C_{ffs}
561	S2	Now we've got it.	C_{ffs}
562	S1	Two times negative six.	C_{ffs}
563	S2	Yeah.	C_{ffs}
565	S2	What's that? It's …	C_{ffs}
566	S1	Plus two. x …	C_{ffs}
567	S2	Six maybe?	C_{ffs}
568	S1	What is it?	C_{ffs}
		Task 5	
575	S1	Was it that you should take … it should be two that you times to six and two that you add to five. Was that it? So it's three and two?	R_{fi}
576	S2	Yeah.	R_{fi}
577	S1	How do we do it? Three plus …	B_{fi}
578	S1	We can test this to see what happens. It was something like this.	B_{fi}
579	S2	Yeah.	B_{fi}
580	S1	It should be 3x and then 2x and then it becomes 5x. Hmm …	C_{ffs}
581	S2	But that's not it, right?	C_{ffs}
582	S1	Let's test it. We should have 3x … plus 6 …	C_{ffs}
585	S1	Two which add to five and then are eight.	C_{ffs}
586	S2	Will something be negative? No …	C_{ffs}
592	S1	Which is it … five or eight?	C_{ffs}
598	S1	But I don't think this works. I don't think we can factor it. But we can try.	C_{ffs}
599	S2	No, I don't think it works.	C_{ffs}
603	S2	Then it should check c, right?	C_{ffs}

604	S1	Yeah, it should check … c.	C_{ffs}
605	S2	It uses c there.	C_{ffs}
606	S2	And if there's a sum that's b.	C_{ffs}
632	S1	… b and then c, and the same with zero also.	C_{ffs}
633	S1	And then we print even in factored. And then it was …	C_{ffs}
634	S1	It's … wait, we don't save them …	C_{ffs}
		Task 6	
665	S1	Okay. So negative 2, right?	B_o, C_{ffs}
666	S2	Yes.	B_o, C_{ffs}
667	S1	x can be equal to 3 … hmmm … can be equal to …	B_o, C_{ffs}
668	S2	Nothing more, right?	B_o, C_{ffs}
669	S1	What else can it be?	B_o, C_{ffs}
670	S2	I don't know.	B_o, C_{ffs}

693	S1	So it's the just the same as it already says, I think.
		`int solution_first_pair = - f.first_pair;`
		`int solution_second_pair = -f.second_pair;`
			C_{ffs}

697	S1	We have to show our calculations … we can write equal to zero here so we actually solve something.
		`printf("(%d + %d)(%d %d)=0\n",`
		`solution_first_pair, f.first_pair,`
		`solution_first_pair, f.second_pair`
			C_{ffs}

699	S1	Now we've got the first answer. We have to do the same thing with the other one.	C_{ffs}
700	S2	It's exactly the same there.	C_{ffs}
714	S1	…+1 … how? It shows … okay, we should maybe show that it … and show factored form like we did earlier.	C_{ffs}
715	S2	Yeah.	C_{ffs}

716	S1	That's a lot easier. Lets do it.
		`x^2 + -5x + 4 = 0`
		`(x + -1)(x + -4)`
		`(1 + -1)(1 + -4) = 0`
		`0 * -3 = 0`
		`(4 + -1)(4+ -4) = 0`
		`3 * 0 = 0`
			C_{ffs}

725	S1	It, like, matches that one. x minus one … x … four. Okay, it finds one solution. One here. And equal to zero. Okay. Four plus one. It finds all solutions here.	C_{ffs}
726	S2	Yeah.	C_{ffs}
740	S1	Isn't it just x plus 1 …	B_e
741	S2	I think so.	B_e
745	S1	No, it's plus one here. Should we times … no, it doesn't work. Or is it just with zero here? Is it x plus two and then zero?	B_e
758	S1	Aha. Um … okay, wait … aha … oy, okay. It's zero, it's not one.	C_{ffs}
759	S2	Yeah.	C_{ffs}
783	S2	If c is equal to zero …	C_{ffs}
784	S1	The we just write …	C_{ffs}
785	S1	It's times x here …	C_{ffs}
787	S1	And it's b here.	C_{ffs}
788	S2	Yeah.	C_{ffs}
789	S1	I think …	C_{ffs}
790	S2	Ah … yeah … that's it.	C_{ffs}
Final task on paper			
838	S1	Three and four work … do they work?	C_{ffs}
839	S2	No …	C_{ffs}
841	S2	It doesn't become negative four.	C_{ffs}
842	S1	No … 1 and 12 don't work … is there something else which?	C_{ffs}
844	S1	Six and two! That's it! Six and two.	C_{ffs}
845	S2	Yeah.	C_{ffs}

861	S1	Negative 2x and plus 6x … isn't that …	C_{ffs}
862	S2	Ah, no …	C_{ffs}
863	S1	That's not it, no … or, yeah, that is it.	C_{ffs}
864	S2	But is it … should it be negative four or just four?	C_{ffs}
872	S1	Oh, right. Yeah, okay, I get what … okay. Aha. It's not plus here, it's negative here. Aha.	C_{ffs}
873	S2	Yeah.	C_{ffs}
874	S1	Okay, okay. Negative six and then …	C_{ffs}
875	S2	Positive two …	C_{ffs}
886	S1	Or not negative 12, negative 2. Like that.	C_{ffs}
887	S2	But wasn't it negative sex and two?	C_{ffs}
888	S1	But that's how it is … the answer is x is ….	C_{ffs}
889	S2	Oh, yeah. Okay, I understand.	C_{ffs}
941	S1	Okay, it's something which times to 12 but adds to seven.	C_{ffs}
942	S2	Yeah.	C_{ffs}
943	S1	Okay, three and four.	C_{ffs}
944	S2	Yeah, its' …	C_{ffs}
945	S1	Is it positive?	C_{ffs}
946	S2	Yeah.	C_{ffs}
953	S1	Okay, then the answer is negative three and negative four.	C_{ffs}
954	S2	Yeah.	C_{ffs}

A2.1.7 C_{st}: Connection Between Standard Form and Solutions

The following utterances show the epistemic actions which lead to C_{st}.

Line	Name	Utterance	Action
		Task 3	
452	S1	Here we can … aha, du you know what we can do?	R_e, R_{mb}
453	S1	These two numbers which we calculate here have to be reused as … or, to get it in standard form … because it's not the same as here even if it looks like it. Or we can just ignore them because they are just negative and just write here …	B_e, B_{mb}, B_{ffs}
456	Both	Notes - The following is a description of what the students were doing based on trying to decipher a few mumbled phrases while looking at the code they were writing: The students are coding a print statement to test what to do with the solutions to create a standard form equation. While coding they realize that the first term is always x^2 and the second is always something times x. They start to think about the connection between the solutions and the coefficients. They start by creating an expanded equation that you get by multiplying out the two factors. They also realize the product is the product of the negative of the solutions. `printf("x^2 + %dx +%dx + %d\n",fy,fx,fy*fx);`	C_{st}
458	S1	Right, it should also actually put those two together. `printf("x^2 + %dx + %d\n",fx+fy,fy*fx);`	C_{st}
470	Me	How did you get coefficient for the middle term?	
471	S2	It's just adding them together.	C_{st}
472	Me	And the last term?	
473	S2	We multiply them with each other.	C_{st}
		Task 4	
523	S2	Now we get the right sum anyways.	B_{fi}
539	S1	It has to … times …	B_{fi}, C_{st}
540	S2	Yeah. I kind of think …	B_{fi}, C_{st}
541	S1	Aha. It should be added together to 13 but times to …	B_{fi}, C_{st}
542	S2	To 12.	B_{fi}, C_{st}
543	S1	Four, three. No. It should …	B_{fi}, C_{st}

551	S2	Yeah, it's 1 and 12.	B_{fi}, C_{st}
552	S1	Aha. I thought it should add to 12, but it should times to 12.	B_{fi}, C_{st}
553	S2	Yes, and then … times to 12 … and …	B_{fi}, C_{st}
561	S2	Now we've got it.	C_{st}
562	S1	Two times negative six.	C_{st}
563	S2	Yeah.	C_{st}
565	S2	What's that? It's …	C_{st}
566	S1	Plus two. x …	C_{st}
567	S2	Six maybe?	C_{st}
568	S1	What is it?	C_{st}
Task 5			
575	S1	Was it that you should take … it should be two that you times to six and two that you add to five. Was that it? So it's three and two?	R_{fi}
576	S2	Yeah.	R_{fi}
577	S1	How do we do it? Three plus …	B_{fi}
578	S1	We can test this to see what happens. It was something like this.	B_{fi}
579	S2	Yeah.	B_{fi}
580	S1	It should be 3x and then 2x and then it becomes 5x. Hmm …	C_{st}
581	S2	But that's not it, right?	C_{st}
582	S1	Let's test it. We should have 3x … plus 6 …	C_{st}
585	S1	Two which add to five and then are eight.	C_{st}
586	S2	Will something be negative? No …	C_{st}
592	S1	Which is it … five or eight?	C_{st}
598	S1	But I don't think this works. I don't think we can factor it. But we can try.	C_{st}
599	S2	No, I don't think it works.	C_{st}
603	S2	Then it should check c, right?	C_{st}
604	S1	Yeah, it should check … c.	C_{st}

605	S2	It uses c there.	C_{st}
606	S2	And if there's a sum that's b.	C_{st}
632	S1	… b and then c, and the same with zero also.	C_{st}
633	S1	And then we print even in factored. And then it was …	C_{st}
634	S1	It's … wait, we don't save them …	C_{st}
Task 6			
647	S1	We should find two which times to negative six but add to negative one.	C_{st}
648	S2	Yeah.	C_{st}
651	S1	I wonder what is actually fastest without a program like this? What is the best way otherwise? [*here they are using their factoring program, which includes the sum of factors in the output. They are starting to wonder how they would do it by hand efficiently*]	B_{fi}, Cst
652	S2	Yeah.	B_{fi}, C_{st}

693	S1	So it's the just the same as it already says, I think.
		`int solution_first_pair = - f.first_pair;`
		`int solution_second_pair = -f.second_pair;`
			C_{st}

697	S1	We have to show our calculations … we can write equal to zero here so we actually solve something.
		`printf("(%d + %d)(%d %d)=0\n",`
		`solution_first_pair, f.first_pair,`
		`solution_first_pair, f.second_pair`
			C_{st}

699	S1	Now we've got the first answer. We have to do the same thing with the other one.	C_{st}
700	S2	It's exactly the same there.	C_{st}
714	S1	…+1 … how? It shows … okay, we should maybe show that it … and show factored form like we did earlier.	C_{st}
715	S2	Yeah.	C_{st}

716	S1	That's a lot easier. Lets do it.
		`x^2 + -5x + 4 = 0`
		`(x + -1)(x + -4)`
		`(1 + -1)(1 + -4) = 0`
		`0 * -3 = 0`
		`(4 + -1)(4+ -4) = 0`
		`3 * 0 = 0`
			C_{st}

725	S1	It, like, matches that one. x minus one … x … four. Okay, it finds one solution. One here. And equal to zero. Okay. Four plus one. It finds all solutions here.	C_{st}
726	S2	Yeah.	C_{st}
740	S1	Isn't it just x plus 1 …	B_e
741	S2	I think so.	B_e
745	S1	No, it's plus one here. Should we times … no, it doesn't work. Or is it just with zero here? Is it x plus two and then zero?	B_e
758	S1	Aha. Um … okay, wait … aha … oy, okay. It's zero, it's not one.	C_{st}
759	S2	Yeah.	C_{st}
783	S2	If c is equal to zero …	C_{st}
784	S1	The we just write …	C_{st}
785	S1	It's times x here …	C_{st}
787	S1	And it's b here.	C_{st}
788	S2	Yeah.	C_{st}
789	S1	I think …	C_{st}
790	S2	Ah … yeah … that's it.	C_{st}
Final task on paper			
838	S1	Three and four work … do they work?	C_{st}
839	S2	No …	C_{st}
841	S2	It doesn't become negative four.	C_{st}
842	S1	No … 1 and 12 don't work … is there something else which?	C_{st}
844	S1	Six and two! That's it! Six and two.	C_{st}
845	S2	Yeah.	C_{st}
861	S1	Negative 2x and plus 6x … isn't that …	C_{st}
862	S2	Ah, no …	C_{st}
863	S1	That's not it, no … or, yeah, that is it.	C_{st}
864	S2	But is it … should it be negative four or just four?	C_{st}

872	S1	Oh, right. Yeah, okay, I get what … okay. Aha. It's not plus here, it's negative here. Aha.	C_{st}
873	S2	Yeah.	C_{st}
874	S1	Okay, okay. Negative six and then …	C_{st}
875	S2	Positive two …	C_{st}
886	S1	Or not negative 12, negative 2. Like that.	C_{st}
887	S2	But wasn't it negative sex and two?	C_{st}
888	S1	But that's how it is … the answer is x is ….	C_{st}
889	S2	Oh, yeah. Okay, I understand.	C_{st}
895	S1	Do we have to … should you rearrange this so it's equal to zero? Or should you …	B_e
896	S2	Yeah.	B_e
898	S1	Or wait. Doesn't it work … can't we just …	B_e
899	S1	But now it's not equal to zero.	B_o, B_e
900	S2	No.	B_o, B_e
928	S1	That we … it was equal to zero.	R_o
929	S2	Yeah.	R_o
930	S1	We could use …	R_o
933	S1	Butt hen … if you could rearrange so in some way … in some way it's equal to zero. Or change it so its equal to zero …	R_e, B_e
934	S2	I can't figure out anyway to …	R_e, B_e
936	S1	But …	R_e, B_e
937	S1	Yes, it's like this.	R_e, B_e
941	S1	Okay, it's something which times to 12 but adds to seven.	C_{st}
942	S2	Yeah.	C_{st}
943	S1	Okay, three and four.	C_{st}
944	S2	Yeah, its' …	C_{st}
945	S1	Is it positive?	C_{st}
946	S2	Yeah.	C_{st}

953	S1	Okay, then the answer is negative three and negative four.	C_{st}
954	S2	Yeah.	C_{st}

A2.2 Coded Lines from Intervention 3 Group 1

The following section list the epistemic actions identified in group 1 in the third intervention.

A2.2.1 C_{fi}: Factors of an Integer

The following utterances show the epistemic actions which lead to C_{fi}. Recognizing and building with actions labeled R_{fi} and B_{fi} refer to previous versions of this construct.

Line	Name	Utterance	Action
Task 1 on paper			
1	S2	We've done this.	R_{fi}
2	S1	Yeah. We didn't do a complete solution.	R_{fi}
6	S1	1 times 6, 2 times 3. But this is what we said. That the square root of 6 is …	B_{fi}
7	S2	Mm-hmm	B_{fi}
8	S1	It's a little more than 2, that's why it turns around here.	B_{fi}
9	S2	Mm-hmm	B_{fi}
10	S1	4 isn't included. 6 times 1 and then … so it's … exactly, that's it.	B_{fi}
11	S1	And then negative is the same, but it's minus 1 times minus 6.	B_{fi}
12	S2	But it also continues down here.	B_{fi}
13	S2	… wait … no.	B_{fi}
14	S1	So it's the same thing for minus … and it turns around here.	B_{fi}
15	S1	If it had been minus 6 … we used like the same thing. Minus 1 times 6. Minus 2 times 3. Minus 3 times 2. Minus 6 times 1. Yes.	B_{fi}
16	S2	It's the same but switched.	B_{fi}
20	S1	Okay, yeah, you can divide it up in prime factors in all the ways you can combine them.	R_p, B_{fi}
30	S1	12 … 4 times 8. 5 doesn't work. 6 works, right?	B_{fi}
31	S2	Times 6.	B_{fi}

33	Me	You are saying “it works”, “it doesn’t work”. What do you mean?	
34	S1	That the number can’t be multiplied by another even number.	B_{fi}
Task 1 on computer			
51	S2	We can do that, but it will be quite hard since we don’t know how far to go. We need to go to the square root of the number. Or the one under it. We don’t know which number that is.	C_{fi}
58	S2	But what happens if the number if negative?	R_p
66	S1	You take the square root of the number so it’s not an integer. Not usually.	R_p
67	S2	Yeah.	R_p
89	S1	But then it’s hard since you need to have some kind of range to know …	B_{fi}
91	Me	How far should you go?	
92	S2	One to …	B_{fi}
93	Me	When should you stop here?	
94	S2	To the number.	B_{fi}
96	S1	The square root of the number.	C_{fi}
97	Me	Try it.	
98	S2	Yeah, the square root of the number.	C_{fi}
117	S2	If h …	C_{fi}
118	S1	… times a number is 6.	C_{fi}
119	S2	Yes, exactly.	C_{fi}
120	S1	Then we can test … we used percent one time …	C_{fi}
135	S1	No, we’ve written number times 2. The number is what the person writes in.	B_{fi}
136	S2	It’s number divided by … 12 … yes, there.	B_{fi}
137	S1	Yes.	B_{fi}
139	S1	So number divided by 2 … times 2. Right? *Changes to* `print(tal/2)`	B_{fi}
146	S1	No, we want it to calculate it. *Changes to* `print((tal/2),"2")`	B_{fi}
147	S2	Yeah.	B_{fi}

A2.2.2 C_e: Understanding What an Equation Is

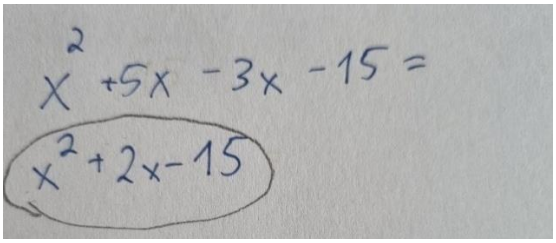
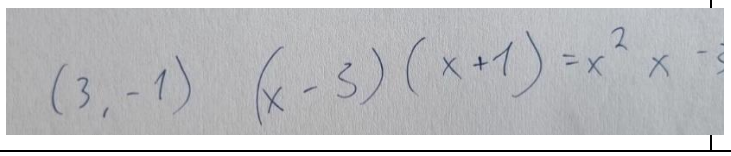
The following utterances show the epistemic actions which lead to C_e.

Line	Name	Utterance	Action
Task 5 on computer			
645	S1	No, it should be minus x.	B_{ffs}, B_{mb}
646	S2	Like this. *Changes the code to* `(r1-r2,r1*r2)`	B_{ffs}, B_{mb}
653	S1	Can't we just add a minus in front?	B_{ffs}, B_{mb}
655	S1	Because that's positive and that's negative. So maybe we should write minus r1? *Change the code to* `(-r1+r2, r1*r2)`, *tests it and sees it doesn't work.*	B_{ffs}, B_{mb}
656	S1	No.	B_{ffs}, B_{mb}
659	S1	It's always negative, so we can have parentheses around it, right? *Change the code to* `(-(r1+r2), r1*r2)`	C_e
Task 6 on computer			
749	S1	It depends on if the number they say is positive or negative. Or yeah, it should be changed, so it has to be minus since that changes it.	C_e

A2.2.3 C_{mb}: Multiplying Two Binomials

The following utterances show the epistemic actions which lead to C_{mb}. Recognizing and building with actions labeled R_{mb} and B_{mb} refer to previous versions of this construct.

Line	Name	Utterance	Action
Task 5 on paper			
490	S1	.. minus 3x ..	R_e, B_e
491	S2	We don't have a rule we can follow here ... minus 15 ... and then minus 3x.	R_e, B_e
494	S1	How were you thinking with minus 15 first?	R_e, B_e
495	S2	Wait, wait.	R_e, B_e
496	S1	It's 5x there.	R_e, B_e

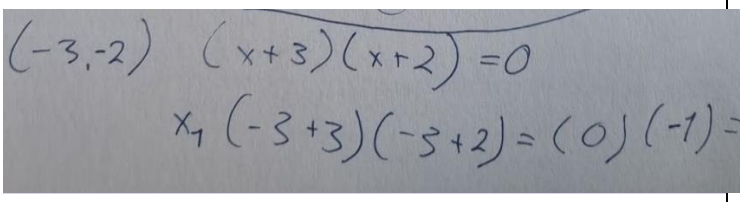
500	S1	And it's x squared plus minus … plus x minus 15 … 	R_e, B_e
Task 5 on computer			
665	S2	… x … minus 3x … that's right.	C_{mb}
666	S1	Yes … and then minus 3. That works. *Checking another row of their output to see if the program works.* 	C_{mb}

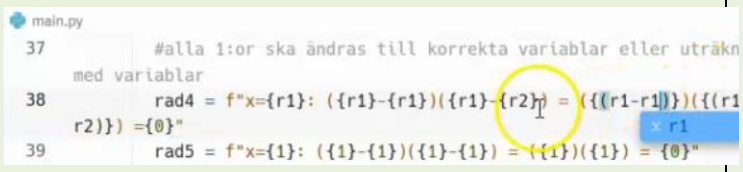
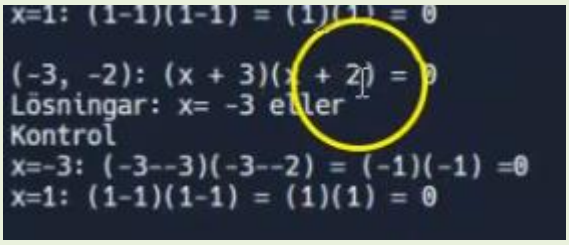
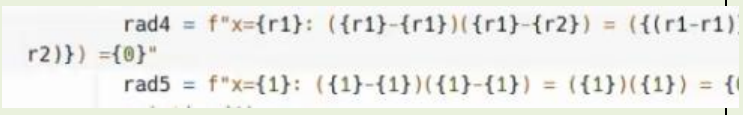
A2.2.4 C_{ffs}: Connection Between Factored Form and Solutions

The following utterances show the epistemic actions which lead to C_{ffs}.

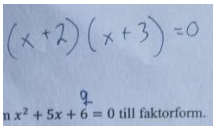
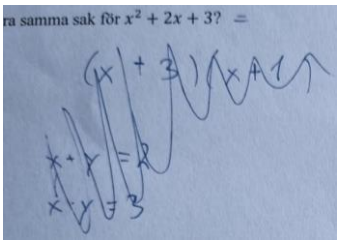
Line	Name	Utterance	Action
Task 3 on computer			
254	S1	It makes sense that it's zero since once of the parentheses is zero.	R_o
255	S2	Yeah.	R_o
Task 4 on paper			
264	S2	x1 is 2. x2 is minus 3.	B_o
265	S1	Yes, exactly.	B_o
267	S1	One parentheses has to be 0. Then it's 0.	B_o
269	S2	One of them can be 2. That can be minus 3. If both are minus 3 then the parentheses is 0. And then the whole product is 0.	C_{ffs}
272	S2	This is harder. That one is much easier to see. *They are comparing $x^2 + 3x - 2x - 6 = 0$ to $(x - 2)(x + 3) = 0$*	C_{ffs}
274	S2	This isn't right since x should be 0 or minus 1 here.	R_o, B_o
275	S1	No, but it should be 6 here.	R_o, B_o

276	S2	Right, right.	R_o, B_o
		Task 4 on computer	
284	S1	Somehow we need to figure out if one of the parentheses are 0.	R_o
291	S1	We need to figure out if one ... when one of the parentheses is 0, then that is a solution.	R_o
293	S1	So this has to be ... since this is minus it has to be the same as that one but positive.	C_{ffs}
294	S2	Yes.	C_{ffs}
295	S1	It's the same, but negative.	C_{ffs}
296	S2	Yes.	C_{ffs}
309	S1	Because it's also that this is minus 3 and this is plus 3. Or something like that. I think. *Trying to understand factored form and use of (x-r1) as standard*	R_{ffs}
310	S2	Ah.	R_{ffs}
311	S1	That's what it's for. It has to calculate this. But ... we need parentheses in some way.	B_o
322	S1	He wants the solutions not the range. *Change code to* `f"x={r1} eller"` *then switch to working on another part.*	B_o, B_e
326	S2	But then it still needs to replace x when we print ... the variable we print out has to be r1.	B_o, B_e
331	S2	We've done it here.	R_{ffs}
332	S2	I'm thinking of exactly this, but instead of x we write r1. That's what I think.	R_{ffs}
358	S1	But this doesn't work. Minus 4 times minus 5 isn't 0. *Output from their test:* `x=-3: (-3-2)(-3-1)=(-5)(-4)=0`	B_o
359	S2	No, that's true.	B_o
360	Me	Maybe it's not always 1 and 2? *Here I am directing their attention to them having hardcoded values.*	B_o
361	S1	No.	B_o
362	S2	How do we do it then? It has to be 0.	B_o
370	S2	But the first part is correct. The one on top. That was the right answer. It should be minus 3 and 0 which is put in here. *They are now looking at the algebra expressions in the first line of the output and thinking about what the calculations should look like based on the algebra.*	R_e

371	S1	Ah, it's 0 there. Yeah, exactly. *Realizing that zero is actual one of the solutions being tested in that row.*	R_e
377	S1	That's plus 2 ... minus 2 .. that one is 1.	B_e
378	S2	Minus 2 is here. Minus 2 minus 2. That's a 1.	B_e
379	S1	And that's a 1 and that's a 2. Positive there and negative there. Can't it be something like that?	B_e
380	S2	Ah. Because it was always minus in there. Minus minus ...	B_e
381	S1	Okay.	B_e
382	S2	So we change all twos to ... to r1 ... change all twos and ones to r1.	B_e
391	S1	Now it's right here. Zero times zero is zero.	R_o, B_o
392	S2	Yes.	R_o, B_o
393	S1	But one times one is not zero. *Referring to a line in the code they haven't edited yet.*	R_o, B_o
394	S2	No, it's not.	R_o, B_o
398	S1	That's not right. 4 times 4 is not 0.	B_o
407	S1	So, one parentheses is zero.	B_o
413	S2	They come from there. That two should go in there. That two should go in there. And it's minus three which goes in here. Where the formula ... *Looking at the equation in factored form and trying to reason to what should be in the calculation output.*	B_p, B_e, B_o
424	Me	Good. What's the solution to the equation?	
425	S1, S2	Minus three and minus two.	C_{ffs}
426	S1	So, it's those in parentheses.	R_{ffs}
427	Me	Show that one of them is a solution. How would you do it? *They check their answer with pen and paper.*	
429	S1	It's zero times minus one is equal to zero. 	C_{ffs}
448	S1	It has to be like this. Look. r2 ... we thought it should be r1 here. Then we wrote r1 here ... it has to be r1 here also. *Indicating the first* `({r1-r2})`	C_{ffs}

449	S1	It has to be r1 here since that's the number we have. Look, there's r1 [*indicating the* `x+3`] and that is also r1 [*indicating the* `(-3,-2)`]. That's r2 and that's r2.	C_{ffs}
450	S2	But before the colon, that's r1. In this case it's 3. Minus 3. It goes there. That's r1. And that's r1. *All of this is trying to explicitly understand how to use the variables in the code to show the connection between factored form and the solutions.*	C_{ffs}
456	S1	But this isn't correct. Look, minus 3 minus minus 3 is zero and here it says minus 1. *Here the student realizes they changed the output of what to calculate but didn't change in the actual calculation.*	B_e
459	S2	And it's still r2 here. 	B_e
460	S1	But it should be that since it's r2 there. 	B_e B_o
461	S1	That's r2, that's r1. 	B_e B_o
468	S2	Is it the same here? Are they the same, but the other way around? So it now says r2 here?	B_e
469	S1	Yes, exactly. I think it should be like that. And it should say r2 here.	B_e
481	S1	Then x has to be equal to ... if x is r2 here, that one, then it will be r2 ...	C_{ffs}
482	S2	Minus r2.	C_{ffs}
483	S1	Minusr1, since it's still 3.	C_{ffs}
484	S2	Right.	C_{ffs}
Task 6 on paper			
668	S1	Factored form was when it was divided up like that.	C_{ffs}
669	S2	Right.	C_{ffs}

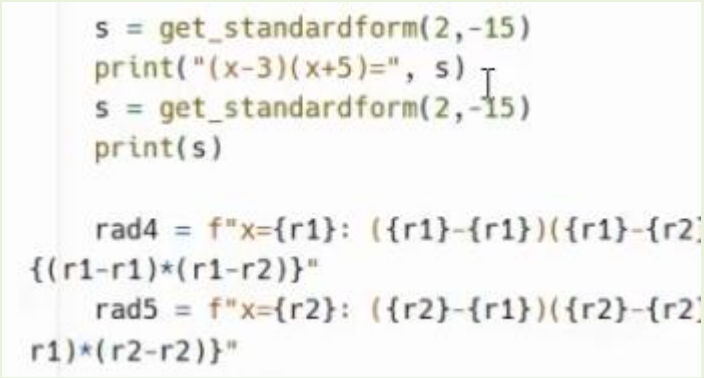
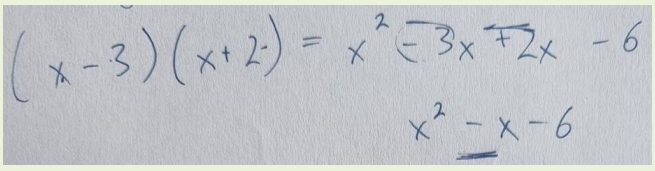
670	S1	Like that … but that's not so hard. One of the parentheses should be zero for each. It's not so hard.	B_o
671	S2	x is 3 … aha … x … x minus 3 … and x minus 4.	C_{ffs}
696	S2	… x plus 3 … x minus 1 … zero	C_{ffs}
Task 6 on computer			
709	S1	I think we should do it like this … it asks for an integer, that's the first one, so it's x minus plus the first integer times x plus minus the second integer.	C_{ffs}
712	S2	Now they are … the two number you get now are r1 and r2.	C_{ffs}
713	S1	Yeah, so we can just change the question marks. *Referring to the question marks in the sample code that they need to edit.*	C_{ffs}
719	S1	Aha. Okay. But that's not so hard either. You just change their signs.	C_{ffs}
720	S2	Yeah.	C_{ffs}
727	S1	That's true. But r1 is a solution, so we should write … wait, every time we want to have … we want to have this form. But then we should write …	C_{ffs}
728	S2	But these are r1 and r2.	C_{ffs}
729	S1	Exactly. Actually, not at all. That's minus r1. *Here it seems the programming is both helping and hurting. It's forcing them to reflect on details, but it is also causing confusion.*	C_{ffs}
748	S2	It should be x minus r1 or plus r1 … x … minus r1.	C_{ffs}
749	S1	It depends on if the number they say is positive or negative. Or yeah, it should be changed, so it has to be minus since that changes it.	C_{ffs}
Task 7 on paper			
789	S2	What are we supposed to do on the paper?	R_{mb}, R_e
790	S1	We're going to factor it.	R_{mb}, R_e
796	S1	Yes, exactly. So the three you take times each other. So it has to be like minus 1 times 3. Minus 1 and 3. Another one works as well. *Here we see a small gap in their understanding when the student suggests there could be more factor pairs of the constant who add up to the middle coefficient.*	B_{mb}, B_e
797	S2	Minus 1 and 3, x plus 1 … x minus …	B_{mb}, B_e
802	S1	Minus 2, minus 3.	C_{ffs}

803	S2	Okay. *They write it down directly* 	C_{ffs}
805	S1	It's hard to do it by hand with that instead.	C_{ffs}
806	S2	Yeah. *Here they are realizing it is easier to see the solutions from factored form compared to when it is in standard form.*	C_{ffs}
829	S1	If we have minus on all of them like we did with the squaring rule. Then it will be minus on the first, plus on the second. If it's plus on both, it needs to be plus on both.	B_{mb}
830	S2	Right.	B_{mb}
831	S1	If it's minus on both, it's minus on one.	B_{mb}
832	S2	Yeah.	B_{mb}
833	S1	If it's plus on a minus then it disappears … *Here they are trying to remember some rules about multiplying binomials connected to the signs inside. But they don't really know why or fully understand or remember the rules.*	B_{mb}
836	S2	Wait, wait, wait. What about that? It should be minus … plus.	C_{ffs}
837	S1	No, that's not it. It should be 4 there in that case. *They are now realizing that their factoring attempt failed on the example that can't be factored, something they didn't see earlier since they didn't look closely enough.* 	C_{ffs}
Final task on paper			
1136	S1	We need to factor minus 24.	C_{ffs}
1137	S2	Ah, okay.	C_{ffs}
1143	S2	That's it. Minus 8 times 3, that's minus 24. Minus 8 plus 3 is minus 5. That's the first one.	C_{ffs}
1151	S1	x1 is 8 and x2 is 3. *They wrote -3 which is correct.*	C_{ffs}

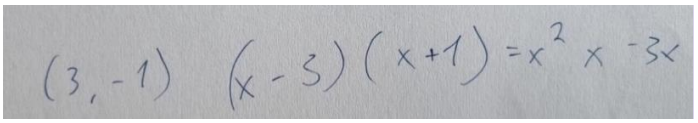
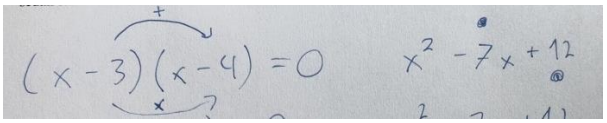
1156	S2	It should be 12 now. *Starting the second problem which has no integer solutions.*	C_{ffs}
1161	S1	Them plus each other is 7, 8, 12.	C_{ffs}
1162	S2	Nothing works.	C_{ffs}

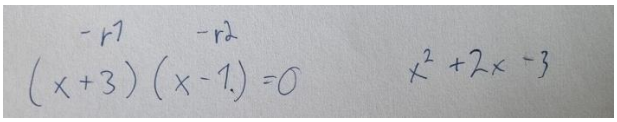
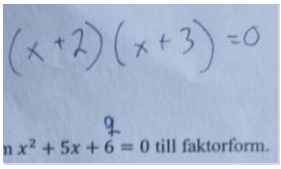
A2.2.5 C_{st}: Connection Between Standard Form and Solutions

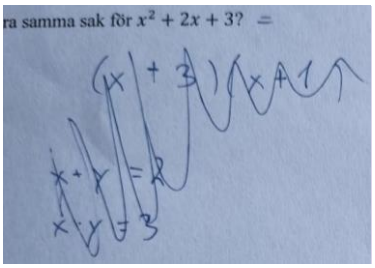
The following utterances show the epistemic actions which lead to C_{st}.

Line	Name	Utterance	Action
Task 5 on computer			
515	S1	I think that 2 and 1 should be changed to something with r.	R_{ffs}
516	S2	Yeah, go for it.	R_{ffs}
517	S1	Should we maybe have r1 here? *They had tested the function with 2 and -15 and were realizing that the coefficients might be related to the solutions.* 	R_{ffs}
541	S1	It's x up to 2, minus 3x .... *They are checking their output by trying to do the calculation in their head to see if it is correct. I suggest they write it down to check it.*	B_{mb}
542	S1	Minus 2x first maybe. We start with ...	B_{mb}
543	S2	And then minus 6. Then it's just minus one x. 	B_{mb}
546	S1	Then it's wrong here.	B_{mb} B_{ffs}

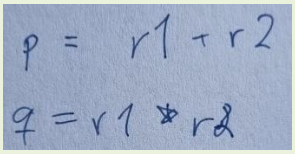
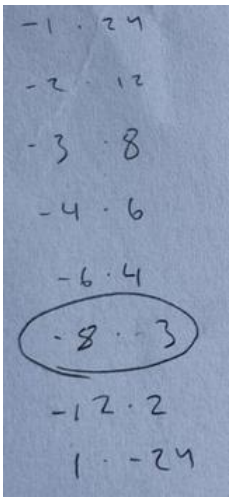
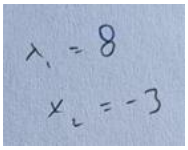
547	S2	Yeah.	B_{mb} B_{ffs}
558	S2	The last number matches in all of them.	B_{mb} B_{ffs}
559	S1	We didn't get a 6 here, we got a 2.	B_{mb} B_{ffs}
560	S2	Oh, right. *Here they are starting to see their error.*	B_{mb} B_{ffs}
561	S2	Wait. The last three ... two, one, zero, zero, minus one, minus one, it just printed the same number ... it didn't multiply.	B_{mb} B_{ffs}
562	S1	We need to somehow get the parentheses to multiply each other. So what have we written that s is equal to here? Get standard form ... we've ... get standard form was here ...	B_{mb} B_{ffs}
563	S1	Get standard form can't be r1 and r2 since they should be the parentheses. Right?	B_{mb} B_{ffs}
564	S2	They were those.	B_{mb} B_{ffs}
565	S1	Because r1 is that one and r2 is that one.	B_{mb} B_{ffs}
566	S2	Yes, since we just took it here, r2, and put it at the end.	B_{mb} B_{ffs}
567	S1	Yeah.	B_{mb} B_{ffs}
576	S1	But it's them times each other. *Referring to the product of the solutions.*	B_{mb} B_{ffs}
577	S2	Yeah. So where have we written how it does s?	B_{mb} B_{ffs}
624	S2	So write r1 times r2 since that is actually it ... in ...	C_{st}
627	S1	That was it. r1 times r2.	C_{st}
628	S2	That was it, yes. Right. *Change code to* `s=get_standardform(r1, r1*r2)`	C_{st}
635	S2	It's actually r1 plus r2.	B_{mb} B_{ffs}
636	S1	Yeah, maybe. *Changes code to* `s=get_standardform(r1+r2, r1*r2)`	B_{mb} B_{ffs}
645	S1	No, it should be minus x.	B_{ffs}, B_{mb}
646	S2	Like this. *Changes the code to* `(r1-r2,r1*r2)`	B_{ffs}, B_{mb}
653	S1	Can't we just add a minus in front?	B_{ffs}, B_{mb}
655	S1	Because that's positive and that's negative. So maybe we should write minus r1? *Change the code to* `(-r1+r2, r1*r2)`*, tests it and sees it doesn't work.*	B_{ffs}, B_{mb}

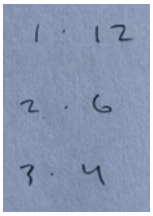
656	S1	No.	B_{ffs}, B_{mb}
659	S1	It's always negative, so we can have parentheses around it, right? *Change the code to* `s=get_standardform(-(r1+r2), r1*r2)`	C_{st}
665	S2	… x … minus 3x … that's right.	C_{st}
666	S1	Yes … and then minus 3. That works. *Checking another row of their output to see if the program works.* 	C_{st}
		Task 6 on paper	
675	S1	And standard form was ax plus …	B_{ffs}, B_{mb}
676	S2	Yes, I think so.	B_{ffs}, B_{mb}
677	S1	… minus 7x …	B_{ffs}, B_{mb}
678	S2	… plus 12 … 	B_{ffs}, B_{mb}
690	S1	So, what did we say? It was that times that.	C_{st}
691	S2	Yeah.	C_{st}
692	S1	It's that …	C_{st}
693	S2	That plus that.	C_{st}
694	S1	No, it wasn't. Or, yes, it is them plus each other and then … it's minus 3 … yes, exactly, minus 7. Them plus each other. *Here they are reflecting on the connection between the solutions and the coefficients in standard form.*	C_{st}
699	S1	So that we connect the solutions and the coefficients in standard form. If we write them down.	C_{st}
700	S2	Yes. It's the same as that one. And that is the same as that … times that.	C_{st}
701	S1	Yeah.	C_{st}
702	S2	Then we can actually write … minus … three … right?	C_{st}

703	S1	Yes. 	C_{st}
Task 6 on computer			
781	S1	We can see if it looks the same … because here … we want to have standard form, right?	C_{st}
782	S2	Yeah. *They copy and paste the arguments from the previous program that they passed to the get standard form function*: `(-(r1+r2),r1*r2)`	C_{st}
Task 7 on paper			
789	S2	What are we supposed to do on the paper?	R_{mb}, R_e
790	S1	We're going to factor it.	R_{mb}, R_e
796	S1	Yes, exactly. So the three you take times each other. So it has to be like minus 1 times 3. Minus 1 and 3. Another one works as well. *Here we see a small gap in their understanding when the student suggests there could be more factor pairs of the constant who add up to the middle coefficient.*	B_{mb}, B_e
797	S2	Minus 1 and 3, x plus 1 … x minus …	B_{mb}, B_e
802	S1	Minus 2, minus 3.	C_{st}
803	S2	Okay. *They write it down directly* 	C_{st}
805	S1	It's hard to do it by hand with that instead.	C_{st}
806	S2	Yeah. *Here they are realizing it is easier to see the solutions from factored form compared to when it is in standard form.*	C_{st}
821	S1	We thought like we did there, that the two together … plus each other should be five and them times each other should be 6. *Here they are explaining their reasoning to me after being asked.*	C_{st}
829	S1	If we have minus on all of them like we did with the squaring rule. Then it will be minus on the first, plus on the second. If it's plus on both, it needs to be plus on both.	B_{mb}
830	S2	Right.	B_{mb}

831	S1	If it's minus on both, it's minus on one.	B_{mb}
832	S2	Yeah.	B_{mb}
833	S1	If it's plus on a minus then it disappears … *Here they are trying to remember some rules about multiplying binomials connected to the signs inside. But they don't really know why or fully understand or remember the rules.*	B_{mb}
836	S2	Wait, wait, wait. What about that? It should be minus … plus.	C_{st}
837	S1	No, that's not it. It should be 4 there in that case. *They are now realizing that their factoring attempt failed on the example that can't be factored, something they didn't see earlier since they didn't look closely enough.* 	C_{st}
Task 7 on computer			
935	S2	Yes! So, in that case the sum … if the sum of something is the same as p, then it should print something.	C_{st}
936	S1	Yes. That was smart.	C_{st}
939	S2	If the sum of something is … is identical with … p … print … it should be something like that. Print … it should print something.	C_{st}
963	S2	We should have "or" in case it's the positive sum or the negative sum.	C_{st}
966	S1	Yeah, if the sum of that, okay, then it is both. Then it has to be both sums, right?	C_{st}
967	S2	Go for it. Go for it.	C_{st}
997	S2	Yes, yes. If the sum and factors of that are that, then it should print.	C_{st}
1023	S1	No, since now it should also be minus.	C_{st}
1024	S2	But then we just write in our "elif".	C_{st}
1029	S1	Then it should print "andragrad" again. *Referring to the variable called "andragrad" which stores the output string for standard form.*	C_{st}

1030	S2	Yeah, I think. Print ”andragrad”
		`if summan12==p:`
		`  print(andragrad)`
		`elif summan34==p:`
		`  print(andragrad)`
			C_{st}

1053	S1	Didn’t we say that in the beginning. I’ve already forgotten what we said. We said it was like r1 and r2.	B_{ffs}
1054	S2	Yeah, right.	B_{ffs}
1070	S2	Then … ahh. Wait, wait, wait. p … p is … what was p? p was that. p is r1 plus r2 and q is r1 times r2. *They were writing down as they were talking* 	C_{st}
1071	S1	Write it in. I think I’m maybe following you.	C_{st}
		Final task on paper	
1136	S1	We need to factor minus 24.	C_{st}
1137	S2	Ah, okay.	C_{st}
1143	S2	That’s it. Minus 8 times 3, that’s minus 24. Minus 8 plus 3 is minus 5. That’s the first one. 	C_{st}
1151	S1	x1 is 8 and x2 is 3. 	C_{st}

1156	S2	It should be 12 now. *Starting the second problem which has no integer solutions.*	C_{st}
1161	S1	Them plus each other is 7, 8, 12. 	C_{st}
1162	S2	Nothing works.	C_{st}

A2.3 Coded Lines from Intervention 3 Group 2

The following section list the epistemic actions identified in group 2 in the third intervention.

A2.3.1 C_0: The Zero Product Property

The following utterances show the epistemic actions which lead to C_0. Recognizing and building with actions labeled R_0 and B_0 refer to previous versions of this construct.

Line	Name	Utterance	Action
Task 3 on computer			
192	S1	Here it goes from minus 7 to minus 12 to minus 15 to minus 16. Then it does down again to 0. Then it goes up. How weird.	B_e
198	S2	I thought as soon as it's … it's 3 and there's it's 5.	R_0, B_0
199	S1	Yes, exactly.	R_0, B_0
200	S2	It's because as soon as it's zero times something, it's zero.	R_0, B_0
201	S1	Yes.	R_0, B_0
202	S3	Exactly.	R_0, B_0
Task 4 on paper			
218	Me	Is there another way to get zero when multiplying two numbers?	B_0
219	S2	No … (*uncertain*)	B_0
220	S1	I don't know. We can test it …	B_0
221	S2	Wait, what did you say?	B_0

222	Me	Is there another way to get zero? If ab=0?	B_o
223	S1	No … (*uncertain*)	B_o
224	S2	No, right? (*uncertain*)	B_o
225	S1	One of them has to be zero.	B_o
226	S2	Yes.	B_o
227	S1	Unless they are the same size. One is like negative 5 and the other is 5. But I don't know f it's possible.	C_o
228	Me	What is it if one is 5 and the other is negative 5?	C_o
229	S1	It's zero.	C_o
230	Me	Is it is zero?	C_o
231	S1	No, it's not.	C_o
232	S2	No, it's not possible.	C_o
233	S1	It has to be 0. Maybe zero, one and ten. No, that's 1.	C_o
234	S2	No, the only way to do it is if one of the factors is zero.	C_o
235	S1	Yes.	C_o

A2.3.2 C_{fi}: Factors of an Integer

The following utterances show the epistemic actions which lead to C_{fi}. Recognizing and building with actions labeled R_{fi} and B_{fi} refer to previous versions of this construct.

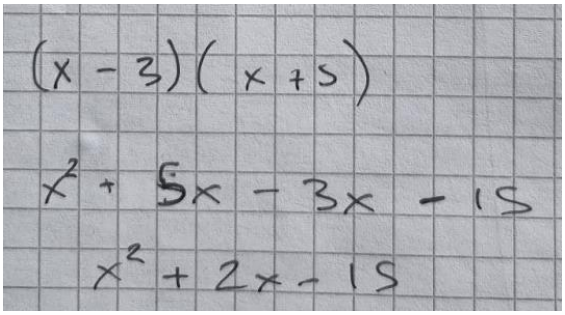
Line	Name	Utterance	Action
Task 2 on computer			
99	Me	But it's not the same factors with minus 6.	
100	S2	Ah, right, for with minus 6 …	R_{fi}
101	S1	Hmm …	R_{fi}
102	S2	Try with a bigger number first to see if it works. Then we can see if it works with negative.	R_{fi}
103	S1	Wait, I have an idea. Take Z divided by Z. That's always 1. Does that work? No, we don't want 1.	R_{fi}

104	S2	Okay, should we find all our negatives?	R_{fi}
105	S1	Wait, it might work … minus Z plus 1. I'll just put parentheses here so it's mathematically correct.	R_{fi}
112	S3	One has to be negative and one has to be positive.	B_{fi}

A2.3.3 C_{mb}: Multiplying Two Binomials

The following utterances show the epistemic actions which lead to C_{mb}. Recognizing and building with actions labeled R_{mb} and B_{mb} refer to previous versions of this construct.

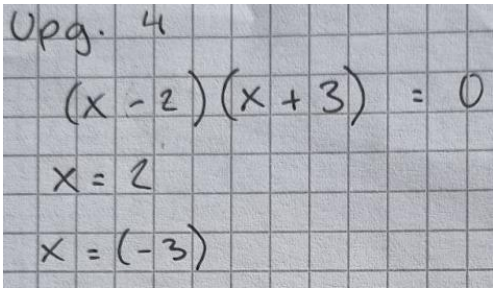
Line	Name	Utterance	Action
Task 3 on computer			
180	S2	If we put in …	R_e
181	S1	Ah, we should put it in.	R_e
182	S2	Yes, the other one …	R_e
184	S2	So, we can write in that there inside parentheses …	B_e
185	S1	We can just take … we start with that.	B_e
186	S2	Yes.	B_e
187	S1	We can write X here.	B_e
188	S2	Yes.	B_e
Task 5 on paper			
536	S1	Do want a specific number, or for all, many numbers?	R_e
542	S2	It's basically simplifying. It's not harder than that …. x minus 3 … x plus 5 … x up to 2 … plus 5x … minus 3x … and then …	C_{mb}
543	S1	Minus 15.	C_{mb}

544	S2	Minus 15. So it's basically x up to 2 plus 2x minus 15. There you have a simplification.  	C_{mb}
545	S1	Did it say minus … wait …	C_{mb}
546	S2	That's a 5.	C_{mb}
547	S3	That's standard form.	C_{mb}
548	S1	Yeah, I think so.	C_{mb}
549	S2	We can simplify factor form to standard form.	C_{mb}
554	S21	Okay, so we multiplied the numbers. And then we got x up to 2 plus 5x minus 3x minus 15. And then vi simplified it. So we got 2x plus …	C_{mb}
555	S2	x up to 2 …	C_{mb}
556	S1	Sorry, x up to 2 plus 2x minus 15	C_{mb}
557	S2	Yes.	C_{mb}

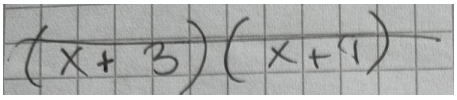
A2.3.4 C_{ffs}: Connection Between Factored Form and Solutions

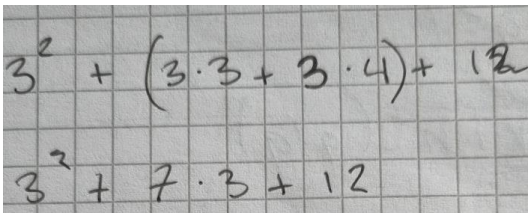
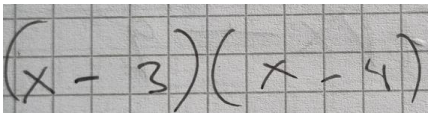
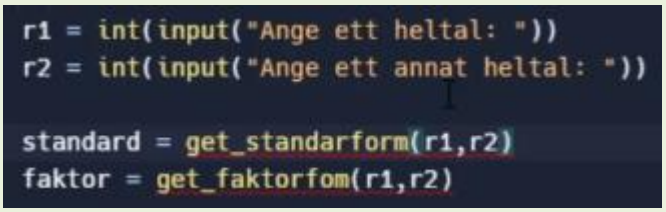
The following utterances show the epistemic actions which lead to C_{ffs}.

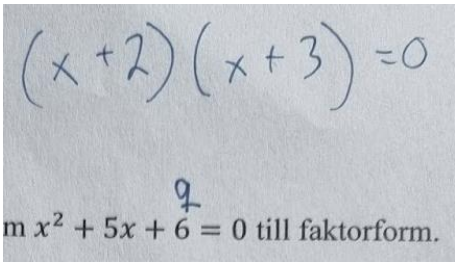
Line	Name	Utterance	Action
Task 2 on computer			
14	S1	Now we need the number.	Re
15	S2	Yes.	Re
16	S1	The number should be a variable.	Re
21	S2	But do we only need one of the factors?	Be
22	S1	No, we need both.	Be
45	S2	z plus x. Or is that hard?	Be
47	S1	I wrote in x here and then z divided by x.	Be

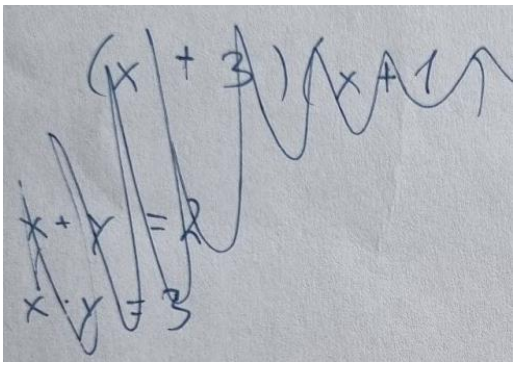
66	S2	Should you change to minus there also?	Be
67	S1	No, it should still be plus. Jag think.	Be
68	S2	I'm thinking that it might be a negative number. Since that's the negative number there.	Be
71	S2	Should that be negative?	Be
Task 3 on computer			
192	S1	Here it goes from minus 7 to minus 12 to minus 15 to minus 16. Then it does down again to 0. Then it goes up. How weird.	B_e
198	S2	I thought as soon as it's … it's 3 and there's it's 5.	R_o, B_o
199	S1	Yes, exactly.	R_o, B_o
200	S2	It's because as soon as it's zero times something, it's zero.	R_o, B_o
201	S1	Yes.	R_o, B_o
202	S3	Exactly.	R_o, B_o
Task 4 on paper			
204	S2	It's easy. Think about 2. If x is 2. If x is minus 3.	R_o
205	S3	Ah, so it's zero.	R_o
206	S1	Either plus 2 or minus 3.	R_o
207	S2	Yes, exactly..	R_o
208	S1	Then it's zero and zero times and it's zero.	B_o
209	S2	Yes, exactly.	B_o
210	S3	So there are two solutions.	B_o
211	S2	There are two solutions.	B_o
212	S1	Zero times everything is just zero.	B_o
213	S2	x is equal to 2 or x is equal to minus 3. 	C_{ffs}
217	S2	If they times each other, if the sum of one of the parentheses is zero, then the whole equations value is zero.	C_{ffs}

Task 4 on computer			
237	S1	We can change it to r1 and r2.	B_o
238	S2	Try it then, r1 and r2.	B_o
263	S2	The solutions are the same as those there. Those are the solutions.	C_{ffs}
266	S1	We should have minus 3 there.	C_{ffs}
267	S2	And minus 1 in the other one.	C_{ffs}
271	S2	We should also write that the solutions are x is equal to minus 3 or x is equal to minus 1.	C_{ffs}
272	S1	Yes, exactly.	C_{ffs}
286	S1	Because that's the opposite, right?	C_{ffs}
287	S2	Yes. It should actually be minus there and plus there. Wait … let me think. Nevermind.	C_{ffs}
344	S3	If we solve this one … the sum of some parentheses has to be 0.	C_{ffs}
345	S2	Yes.	C_{ffs}
346	S3	Because it's the same thing as two factors. So if we can find, or if you know something … it's hard.	C_{ffs}
347	S2	It's really hard … so if the sum is equal to 0 …	C_{ffs}
348	S3	Exactly.	C_{ffs}
414	S1	Wait, a question … is the task that we want everything that is zero?	B_o
415	S2	Yes.	B_o
416	S1	Or do we want everything possible?	B_o
417	S2	Everything that is zero.	B_o
432	S2	x is equal to … r1 ….	C_{ffs}
433	S1	x should be …. okay, let's do it. It should be r1.	C_{ffs}
434	S2	Yes.	C_{ffs}
435	S1	And that one should be …	C_{ffs}
436	S2	Should be r2.	C_{ffs}
437	S1	… r2.	C_{ffs}
438	S2	Yes.	C_{ffs}
439	S1	This one should be … it should be r1 and that should be r2.	C_{ffs}
440	S2	Yes, exactly.	C_{ffs}

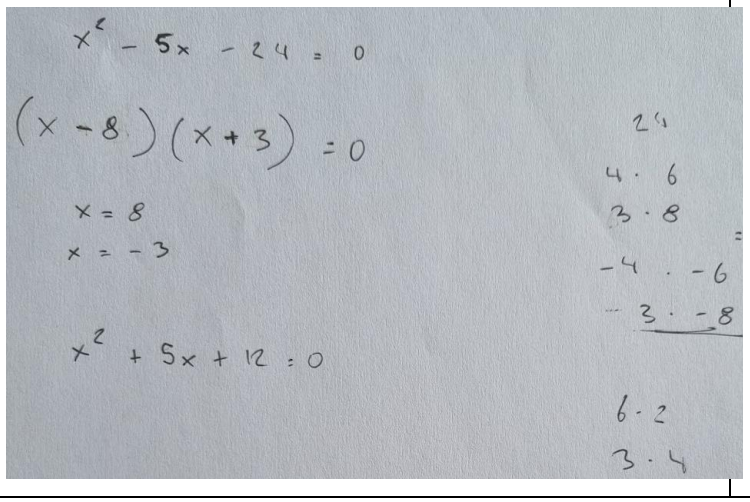
444	S2	r1. And that one is r1. And that one is r2.	C_{ffs}
445	S1	Yes. Then we want the same but we want to change …	C_{ffs}
446	S2	It's r1.	C_{ffs}
447	S1	Yes.	C_{ffs}
483	S2	What the fuck? What happened there? It's 3 minus 3. How is that r1? (3, 3): (x - 3)(x - 3) = 0 Lösningar: Denna text ersätts med lösningarna Kontrol x=3: (3-3)(3-3) = (0)(0) = 0 x=3: (3-3)(3-3) = (1)(1) = 1	C_{ffs}
484	S1	3 minus 3 is …	C_{ffs}
485	S2	Zero.	C_{ffs}
486	S1	It's zero times zero.	C_{ffs}
516	S1	,,, x is equal to r2. So. I think it'll work. (3, -1): (x - 3)(x + 1) = 0 Lösningar: x = 3 eller x = -1 Kontrol x=3: (3-3)(3--1) = (0)(4) = 0 x=3: (-1-3)(-1--1) = (-4)(0) = 0	C_{ffs}
517	S2	Yes.	C_{ffs}
		Task 6 on paper	
688	S1	Factor form, that's those … it's just those there. I don't know.	R_{ffs}
689	S2	No.	R_{ffs}
694	S2	So it's basically … x … plus 3 … x plus 4? 	B_{ffs}
697	S1	We want the answer to be x. The answer should be x is equal to 3.	C_{ffs}
698	S2	x should be equal to 3 and x can be 4.	C_{ffs}
699	S3	Yes.	C_{ffs}
700	S2	So the solutions are x is 3 and … there are two solutions. Then the solution is if x … if x is … there are two solutions.	C_{ffs}
701	S1	Aha.	C_{ffs}

708	S2	Okay, what if we do it like this. If x is 3 then it's 3 up to 2 … plus …  	B_e, B_{mb}
718	S3	So here we have the solutions for x.	C_{ffs}
719	S2	Then it's just to put them in the values. It's x minus 3 … x minus 4. Then we have factor form for it.  	C_{ffs}
720	S1	Yes. That's actually completely correct.	C_{ffs}
721	S3	What did she do?	C_{ffs}
722	S1	She put 3 in there and 4 in there. *indicating where they had earlier written (x-r1)(x-r2)*	C_{ffs}
727	S2	I think they are positive there do they are negative here.	C_{ffs}
728	S1, S3	Yes.	C_{ffs}
		Task 6 on computer	
779	S2	Put in r1 and r2?  	R_{ffs}
781	S2	Ah, that's standard form.	B_{st}, B_{ffs}
782	S1	Standard form … should we write … why is it wrong?	B_{st}, B_{ffs}
784	S1	Okay, now I understand. If we need to have … how the hell am I going to write this?	B_{st}, B_{ffs}
786	S1	That's standard form. Right?	B_{st}, B_{ffs}
787	S2	That's factor form.	B_{st}, B_{ffs}
788	S1	That's standard form.	B_{st}, B_{ffs}
789	S2	That one is standard form.	B_{st}, B_{ffs}

793	S2	So we should write … let's use that number as an example. The program should basically print out x plus 2, x minus 1. It should then write x up to 2 and then … plus x and then minus 2. So basically that should be printed out.	C_{ffs}
794	S1	Yes.	C_{ffs}
795	S2	Then you have standard form there and factor form there.	C_{ffs}
808	S2	I'm pretty sure it should be x minus, say a, x minus b.	C_{ffs}
817	S2	If we write r1 and r2. And something else down there because that's standard form.	B_{st}, B_{ffs}
847	S1	r1 is input … that's factor form.	C_{ffs}
848	S2	Yes.	C_{ffs}
854	S2	But that was standard form. If it's factor form then it's not r1 plus r2. It's x plus r1 and x plus r2. *Confused a bit about the signs, and also how it connects to function input in the program.*	C_{ffs}
870	S2	It has to be r1 … and r2. And up there vi need to use …	C_{ffs}
		Task 7 on paper	
915	S1	If they are multiplied we get that one, if they are added we get that one.	C_{ffs}
916	S2	Then we've got it. 	C_{ffs}
922	S1	I think it's just x and 3 … no … damn. I think it's just 3 and 1.	C_{ffs}
923	S2	How … what?	C_{ffs}
924	S1	No, it's not minus, it's …	C_{ffs}
927	S1	No, it isn't minus. If it was minus it would be negative.	C_{ffs}
928	S2	… plus 3x …	C_{ffs}
929	S1	Yes.	C_{ffs}
930	S2	Minus 1 times x minus one x …	C_{ffs}
931	S1	Yes.	C_{ffs}
932	S2	Minus 1 times 3 …	C_{ffs}

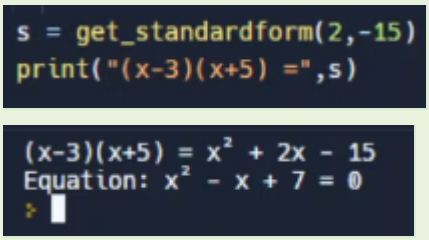
933	S1	It's minus 3. Like I said.	C_{ffs}
937	S1	Both can't be negative because then it would be negative. If both were negative then that one would be negative and that one positive.	C_{ffs}
939	S3	Is it even possible?	C_{ffs}
940	S1	I'm trying to think. I don't think it's possible.	C_{ffs}
941	S2	It's not solvable.	C_{ffs}
947	S1	I don't think so. Because … no, it doesn't work. Because if you had something positive … if you had two positive numbers, then it would be bigger than 2.	C_{ffs}
949	S1	If we add two ones we get … it has to be 1 and 1 and we get 2 there, if we add two numbers without a minus.	C_{ffs}
950	S2	And we need a 3.	C_{ffs}
951	S1	If we have 1 times 1 we get 1. So it doesn't work.	C_{ffs}
953	S1	If one is minus, then that will be minus and it doesn't work.	C_{ffs}
955	S1	If both are minus, then that will also be minus. That will be positive but that will be minus. That's plus, so it doesn't work.	C_{ffs}
956	S3	It's not possible.	C_{ffs}
957	S2	It's not possible. 	C_{ffs}
986	S1	We figured out that if one is negative, then that will be negative.	C_{ffs}
987	S2	Yes.	C_{ffs}
988	S1	And if both are negative, then that will be positive, but that will be negative.	C_{ffs}
989	S2	Yes.	C_{ffs}
990	S1	And if both are positive, then we have to add them … then it has to be 1 and 1. So it has to be 2. And 1 times 1 is 1 so it's not 3.	C_{ffs}
993	S2	The factors of 6 … are 3 and 2. And 3 plus 2 is 5. *Explaining how they did the first one.*	C_{ffs}

Final task on paper			
1025	S2	The factors are there … 24. It's either 4 times 6, 3 times 8. And minus 4, minus 6 and minus 3, minus 8. Their sum is 10. Their sum is 11. The sum of minus 4 plus minus 6 … is that the one we should use?	C_{ffs}
1026	S3	Minus 5.	C_{ffs}
1027	S2	That one … minus 3 plus minus 8 is 5, or it's minus 5.	C_{ffs}
1028	S3	It feels like it should be that one.	C_{ffs}
1029	S2	It's 3, 8. *A number of mistakes here, starting with factoring +24 instead of -24 and then adding negative numbers wrong.*	C_{ffs}
1030	S1	One of them should be negative.	C_{ffs}
1031	S2	Exactly … so it's x … minus …	C_{ffs}
1032	S1	It must be plus minus, right? Plus minus 3.	C_{ffs}
1033	S2	*Writes (x-8)(x-3) which doesn't work.*	C_{ffs}
1034	S1	I thought that could be positive and that could be negative.	C_{ffs}
1035	S2	Does it matter?	C_{ffs}
1040	S2	Yeah, it does. It absolutely matters. *Write (x+8)(x-3)*	C_{ffs}
1045	S1	So that should be minus and that should be plus.	C_{ffs}
1046	S2	Exactly. *Writes (x-8)(x+3)*	C_{ffs}
1050	S3	Here you have x is 8, and here x minus 3.	C_{ffs}
1052	S2	6 times 2 … there's 3 times 4. They all have to be positive because it is always positive. But none of these add up to 5. So it's not possible.	C_{ffs}
1053	S3	Unless there's a negative.	C_{ffs}

1054	S2	No, there isn't a negative because it's all positive. And no sum of factors is 5. So it's not possible.  	C_{ffs}

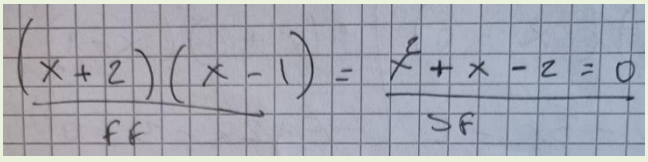
A2.3.5 C_{st}: Connection Between Standard Form and Solutions

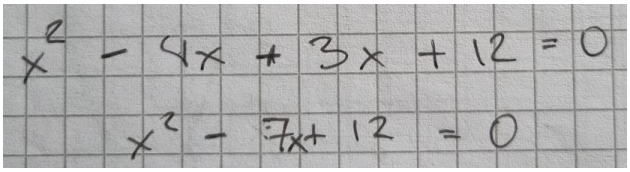
The following utterances show the epistemic actions which lead to C_{st}.

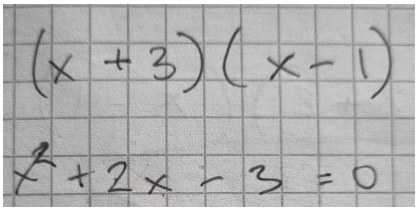
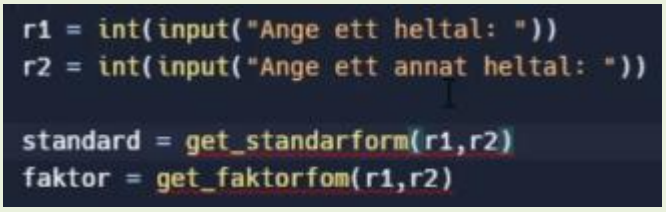
Line	Name	Utterance	Action
		Task 5 on computer	
583	S1	I wonder where 2x came from.  	R_{mb}
584	S2	2x is ... 5x minus 3x is 2x.	R_{mb}
595	S1	The answer goes in here. This is where we put in 2x.	B_{mb}
596	S2	Yes.	B_{mb}
597	S1	So here is 2 and there is 2 and there is minus 15 and there is minus 15.	B_{mb}
598	S2	Yes.	B_{mb}
599	S1	So we took ... we multiplied them. Wait, what did we do? To get 15 we can multiply r1 and r2.	B_{mb}
600	S2	Yes.	B_{mb}
601	S1	That works. Since we get minus 15 like with did here. Right?	B_{mb}

602	S2	Yes. Because … for p you have to take … r1 …	C_{st}
603	S1	It's q … q is r1 times r2. And p …	C_{st}
604	S2	Yes.	C_{st}
608	S2	r1 minus r2	B_{mb}
609	S1	Or add them. Because it's 2, because it's … it's negative. It's addition.	B_{mb}

615	S2	It works for … for some. (*doesn't see the sign error*)
		`s = get_standardform(r1 + r2,r1 * r2)`
		`(2, -1): (x - 2)(x + 1) = x² + x - 2 = 0`
		`Lösningar: x = 2 eller x = -1`
		`Kontrol:`
		`x=2: (2-2)(2--1) = (0)(3) = 0`
		`x=-1: (-1-2)(-1--1) = (-3)(0) = 0`
		`(2, 0): (x - 2)(x) = x² + 2x = 0`
		`Lösningar: x = 2 eller x = 0`
		`Kontrol:`
		`x=2: (2-2)(2-0) = (0)(2) = 0`
		`x=0: (0-2)(0-0) = (-2)(0) = 0`
			C_{st}

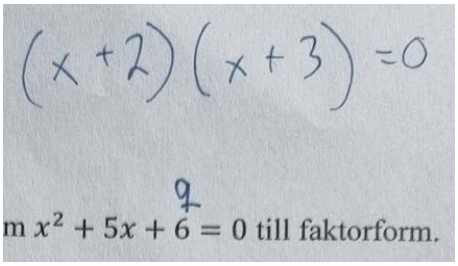
616	S1	The one at the bottom … (*the sign error is hidden in this one*) `(3, -3): (x - 3)(x + 3) = x² - 9 = 0`	C_{st}
624	S2	But then it's right. `(-2, 1): (x + 2)(x - 1) = x² - x - 2 = 0`	B_{mb}
625	S1	Yeah, it's correct. (*missing that it's wrong and doesn't match example in the task*)	B_{mb}
638	S1	No, it should be plus there. It isn't (*in the output*) 	C_{st}
639	S2	Ah.	C_{st}
640	S1	It isn't correct. (*in the program - here they are experiencing cognitive conflict which was the intention of the task*)	C_{st}
658	S2	But use … test minus then.	B_{mb}
659	S1	We should use minus with them.	B_{mb}
660	S2	Use subtraction.	B_{mb}
661	S1	Which one should be minus?	B_{mb}

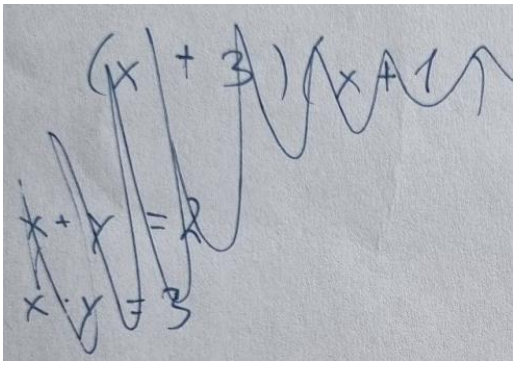
662	S2	Just use minus between them. We can see if it's correct. (*they still don't understand they need to take negative of the same, not the difference*) `s = get_standardform(r1 - r2,r1 * r2)`	B_{mb}
665	S2	It's wrong. It's 5x here. `(3, -2): (x - 3)(x + 2) = x² + 5x - 6 = 0`	C_{st}
666	S1	Yeah, I thought so since if it takes minus 2 minus 1 then it's minus 3. So that doesn't work.	C_{st}
667	S1	But if we make that positive and that negative …	C_{st}
669	S1	I wonder if it's so simple that we just … can you make them negative like that? You can. No, I need to use times.	C_{st}
671	S2	What are you doing?	C_{st}
672	S1	I'm just changing from negative to positive. Or, it depends. `s = get_standardform(-1 * r1 + r2 * -1,r1 * r2)`	C_{st}
		Task 6 on paper	
730	S1	Standard form is x …	B_{mb}
731	S2	x up to 2 …	B_{mb}
732	S1	Plus …	B_{mb}
733	S2	… minus 3 … 	B_{mb}
734	S2	That's standard form.	B_{mb}
751	S1	The connection is that they are multiplied and then the opposite of their … am jag wrong now? No they are just multiplied. Since the one that doesn't have an x in it is multiplied by two …	B_{mb}
752	S3	It has something to do with when we added … 4 plus … or what did we do? When we put those two together?	R_{mb}, R_{ffs}
753	S1	When we put them together … we answered something else. We put those together, those we multiplied.	C_{st}
772	S1	It's 2x and … minus 3.	C_{st}
773	S2	Plus 2x minus 3.	C_{st}

774	S1	Plus 2x minus 3. *Here they used the formula for the coefficients directly, not multiplying out the binomial.* 	C_{st}
		Task 6 on computer	
779	S2	Put in r1 and r2? 	R_{st}
781	S2	Ah, that's standard form.	B_{st}, B_{ffs}
782	S1	Standard form ... should we write ... why is it wrong?	B_{st}, B_{ffs}
784	S1	Okay, now I understand. If we need to have ... how the hell am I going to write this?	B_{st}, B_{ffs}
786	S1	That's standard form. Right?	B_{st}, B_{ffs}
787	S2	That's factor form.	B_{st}, B_{ffs}
788	S1	That's standard form.	B_{st}, B_{ffs}
789	S2	That one is standard form.	B_{st}, B_{ffs}
793	S2	So we should write ... let's use that number as an example. The program should basically print out x plus 2, x minus 1. It should then write x up to 2 and then ... plus x and then minus 2. So basically that should be printed out.	C_{st}
794	S1	Yes.	C_{st}
795	S2	Then you have standard form there and factor form there.	C_{st}
808	S2	I'm pretty sure it should be x minus, say a, x minus b.	C_{st}
817	S2	If we write r1 and r2. And something else down there because that's standard form.	B_{st}, B_{ffs}
823	S2	And then p, q like we did before. Negative in parentheses r1 plus r2.	C_{st}
878	S1	So in standard we should just have p and q. Right?	C_{st}
879	S2	Yes. And that's how we get p and q. P is by multiplying.	C_{st}

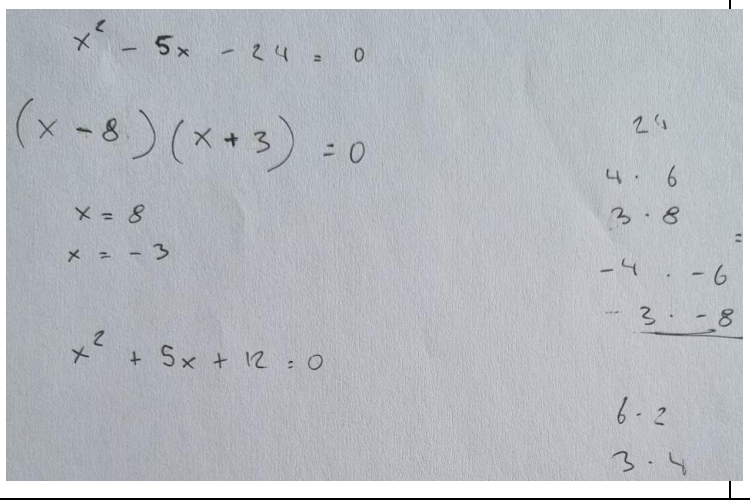
880	S1	Yeah, it is. That's totally correct.	C_{st}
881	S2	So it's that one. That's p.	C_{st}
882	S1	That's correct.	C_{st}

894	S1	Aha. Du did it like that. You used what it says here and it works.
		`r1 = int(input("Ange ett heltal: "))`
		`r2 = int(input("Ange ett annat heltal: "))`
		`standard = get_standardform(-(r1+r2),r1*r2`
		`faktor = get_faktorform(r1,r2)`
			C_{st}

895	S2	Now we are done.	C_{st}
Task 7 on paper			
897	S3	We need to go backwards.	R_{ffs}
898	S2	Yes.	R_{ffs}
902	S3	3 and 2.	B_{ffs}
903	S1	Yeah, it's 3 and 2 for that one.	B_{ffs}
904	S2	Plus or minus 3	B_{ffs}
915	S1	If they are multiplied we get that one, if they are added we get that one.	C_{st}
916	S2	Then we've got it. 	C_{st}
922	S1	I think it's just x and 3 … no … damn. I think it's just 3 and 1.	C_{st}
923	S2	How … what?	C_{st}
924	S1	No, it's not minus, it's …	C_{st}
927	S1	No, it isn't minus. If it was minus it would be negative.	C_{st}
928	S2	… plus 3x …	C_{st}
929	S1	Yes.	C_{st}
930	S2	Minus 1 times x minus one x …	C_{st}

931	S1	Yes.	C_{st}
932	S2	Minus 1 times 3 …	C_{st}
933	S1	It's minus 3. Like I said.	C_{st}
937	S1	Both can't be negative because then it would be negative. If both were negative then that one would be negative and that one positive.	C_{st}
939	S3	Is it even possible?	C_{st}
940	S1	I'm trying to think. I don't think it's possible.	C_{st}
941	S2	It's not solvable.	C_{st}
947	S1	I don't think so. Because … no, it doesn't work. Because if you had something positive … if you had two positive numbers, then it would be bigger than 2.	C_{st}
949	S1	If we add two ones we get … it has to be 1 and 1 and we get 2 there, if we add two numbers without a minus.	C_{st}
950	S2	And we need a 3.	C_{st}
951	S1	If we have 1 times 1 we get 1. So it doesn't work.	C_{st}
953	S1	If one is minus, then that will be minus and it doesn't work.	C_{st}
955	S1	If both are minus, then that will also be minus. That will be positive but that will be minus. That's plus, so it doesn't work.	C_{st}
956	S3	It's not possible.	C_{st}
957	S2	It's not possible. 	C_{st}
986	S1	We figured out that if one is negative, then that will be negative.	C_{st}
987	S2	Yes.	C_{st}
988	S1	And if both are negative, then that will be positive, but that will be negative.	C_{st}
989	S2	Yes.	C_{st}
990	S1	And if both are positive, then we have to add them … then it has to be 1 and 1. So it has to be 2. And 1 times 1 is 1 so it's not 3.	C_{st}

993	S2	The factors of 6 … are 3 and 2. And 3 plus 2 is 5. *Explaining how they did the first one.*	C_{st}
Final task on paper			
1025	S2	The factors are there … 24. It's either 4 times 6, 3 times 8. And minus 4, minus 6 and minus 3, minus 8. Their sum is 10. Their sum is 11. The sum of minus 4 plus minus 6 … is that the one we should use?	C_{st}
1026	S3	Minus 5.	C_{st}
1027	S2	That one … minus 3 plus minus 8 is 5, or it's minus 5.	C_{st}
1028	S3	It feels like it should be that one.	C_{st}
1029	S2	It's 3, 8. *A number of mistakes here, starting with factoring +24 instead of -24 and then adding negative numbers wrong.*	C_{st}
1030	S1	One of them should be negative.	C_{st}
1031	S2	Exactly … so it's x … minus …	C_{st}
1032	S1	It must be plus minus, right? Plus minus 3.	C_{st}
1033	S2	*Writes (x-8)(x-3) which doesn't work.*	C_{st}
1034	S1	I thought that could be positive and that could be negative.	C_{st}
1035	S2	Does it matter?	C_{st}
1040	S2	Yeah, it does. It absolutely matters. *Write (x+8)(x-3)*	C_{st}
1045	S1	So that should be minus and that should be plus.	C_{st}
1046	S2	Exactly. *Writes (x-8)(x+3)*	C_{st}
1050	S3	Here you have x is 8, and here x minus 3.	C_{st}
1052	S2	6 times 2 … there's 3 times 4. They all have to be positive because it is always positive. But none of these add up to 5. So it's not possible.	C_{st}
1053	S3	Unless there's a negative.	C_{st}

1054	S2	No, there isn't a negative because it's all positive. And no sum of factors is 5. So it's not possible.  	Cst

Programming as a Tool for Helping Students Understand and Solve Quadratic Equations

This study explored the idea that working on programming tasks may help students learn mathematics. The aim of the study was to determine if the act of programming itself could help students construct new mathematical knowledge. This was motivated by the idea that in order to be able to write a program to deal with general solutions and methods, the students would need to reflect on their own knowledge and make their reasoning explicit and detailed. Three intervention cycles were conducted with different groups of Swedish secondary school students, the majority of which were novice programmers. The interventions themselves were modeled on task-based interviews though the overarching methodology and analysis was based on the Abstraction in Context theoretical framework. The findings indicate that although it seems possible for the act of programming itself to help students construct new mathematical concepts, it is unlikely to be an effective classroom strategy. The cognitive difficulties novice programmers encountered when writing programs became an obstacle to learning new mathematics as well as dramatically increasing the time needed to complete apparently simple tasks.

ISBN 978-91-7867-502-9 (print)

ISBN 978-91-7867-503-6 (pdf)

ISSN 1403-8099

LICENTIATE THESIS | Karlstad University Studies | 2024:35