Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Anne Margrethe Vestgøte Bosch

Art Production with Programming and Trigonometry

An Experiment in Mathematics 1T According to the Principles of Didactical Engineering

Master's thesis in Natural Science with Teacher Education
Supervisor: Majid Rouhani
June 2022

NTNU
Kunnskap for en bedre verden

Anne Margrethe Vestgøte Bosch

Art Production with Programming and Trigonometry

An Experiment in Mathematics 1T According to the Principles of Didactical Engineering

Master's thesis in Natural Science with Teacher Education
Supervisor: Majid Rouhani
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU

Kunnskap for en bedre verden

Abstract

Computing has in recent years been included in compulsory education in several European countries to equip students with the necessary skills for the 21st-century. In Norway, programming and computational thinking were included in primary and secondary education and training in the renewed National Curriculum in 2020. The approach chosen in Norway is to include programming and computational thinking in mathematics and other existing school subjects. Programming is a powerful medium for expression that has the potential to open up new possibilities particular to the mathematics subject. For example, programming can lead to engagement and a deeper understanding of the mathematical content. However, there are also several challenges associated with the inclusion of programming into existing mathematics subjects, such as the threat of content overload. There is sparse research documenting the effects of different approaches to including programming in existing mathematics subjects. Therefore more research is needed to support the teachers in operationalizing the curriculum.

This study aims to gain insight into and provide knowledge about how a sequence of teaching situations in the subject Mathematics 1T on the upper secondary level in Norway can be designed with the intent to enable knowledge development in both programming and trigonometry. The target knowledge in trigonometry intended was for the students to evaluate and apply trigonometric theorems in problem-solving with programming. The target knowledge intended in programming was for the students to plan and implement subroutines in problem-solving with programming. Through realizing a self-designed sequence of lectures with two Mathematics 1T classes, this study has investigated the contributing and constraining factors for the students' possibilities for obtaining the target knowledge.

The French didactics tradition has inspired the approach to address the research questions in this study. A flexible and qualitative methodology inspired by Didactical Engineering (DE) was applied with the conceptual framework provided by the Theory of Didactical Situations in Mathematics (TDS). First, preliminary analyses of the programming and trigonometry components of the target knowledge were conducted. Then, a sequence of lectures was designed. In line with TDS, the lecture sequence was a didactical situation aimed at teaching the target knowledge, and the design was founded on the preliminary analyses. The didactical situation was piloted and then realized twice with Mathematics 1T classes. The data collected was the Python programs created by the students in the didactical situation, observational data from the realizations, and student interviews.

The results from analyzing the data material indicate that the didactical situation has the potential for students to attain the target knowledge. Factors in the didactical situation contributing toward that end were the PRIMM structure of the assignments, the visual feedback enabled by the Python turtle library, and the pair programming collaboration method. The results also indicate that several factors in the didactical situation constrained the students' progression towards the target knowledge. Among the identified constraining factors were students' unsystematic debugging of errors in their programs and misconceptions about for-loops and the local scope of parameters in subroutines.

This study provides insight into the inclusion of programming in the subject Mathematics 1T. This study has identified contributing and constraining factors for the students' possibilities for obtaining the target knowledge in the designed didactical situation. The insights can be applied to further develop and improve the didactical situation prior to new classroom realizations. Furthermore, this study has shown that the DE research methodology combined with TDS can provide valuable insight into the field intersecting programming and mathematics education. Consequently, the approach taken in this study may inspire future research projects in the field.

Sammendrag

Algoritmisk tenkning og programmering har de siste årene blitt innlemmet i den obligatoriske skolegangen i flere europeiske land for å utruste elevene med den nødvendige kompetansen og ferdighetene som kreves i det 21. århundre. I Norge ble programmering og algoritmisk tenkning inkludert i grunnopplæringen gjennom det nye læreplanverket som kom i 2020, også kalt Fagfornyelsen. Tilnærmingen som ble valgt i Norge var å inkludere programmering og algoritmisk tenkning i matematikk og andre eksisterende skolefag. Programmering er et kraftfullt medium som har potensiale til å skape nye muligheter i matematikkfaget. For eksempel kan programmering føre til engasjement og en dypere forståelse av det matematiske innholdet. Det følger imidlertid også med utfordringer når programmering inkluderes i eksisterende matematikkfag, slik som stofftrengsel. Det finnes lite forskning på effekten av ulike tilnærminger til å inkludere programmering i eksisterende matematikkfag. Mer forskning på dette feltet er derfor nødvendig for å støtte lærerne som operasjonaliserer læreplanene i klasserommene.

Målet med denne studien var å gi innsikt i og kunnskap om hvordan en sekvens av undervisningsopplegg i faget Matematikk 1T på videregående skole i Norge kan utformes med den hensikt å muliggjøre kunnskapsutvikling innen både programmering og trigonometri. Den tilsiktede målkunnskapen innenfor trigonometri var at elevene skulle evaluere og anvende trigonometriske setninger i problemløsning med programmering. Den tilsiktede målkunnskapen innenfor programmering var at elevene skulle planlegge og implementere subrutiner i problemløsning med programmering. Gjennom realisering av undervisningssekvensen i klasserommet med to Matematikk 1T-klasser har denne studien undersøkt de muliggjørende og forhindrende faktorene for elevenes muligheter til å tilegne seg målkunnskapen.

Tilnærmingen til å besvare forskningsspørsmålene i denne studien er inspirert av den franske didaktikktradisjonen. En fleksibel og kvalitativ metode inspirert av didaktisk ingeniørvitenskap (DE) ble brukt sammen med teorien for didaktiske situasjoner i matematikk (TDS) som konseptuelt rammeverk. Den første fasen av studien var forberedende analyser av programmeringskomponenten og trigonometrikomponenten av målkunnskapen. Deretter ble sekvensen av undervisningsopplegg utviklet. I tråd med TDS var sekvensen av undervisningsopplegg en didaktisk situasjon med hensikt å undervise i målkunnskapen, og designet var fundert i de forberedende analysene. Den didaktiske situasjonen ble pilotert og deretter realisert i to matematikk 1T-klasser. Datamaterialet som ble samlet inn bestod av elevenes Python-programmer som ble laget i løpet av den didaktiske situasjonen, observasjonsdata fra realiseringene og elevintervjuer.

Resultatene fra analyse av datamaterialet tyder på at den didaktiske situasjonen har potensial for at elevene kan oppnå målkunnskapen. Faktorer i den didaktiske situasjonen som bidro til dette var PRIMM-strukturen i oppgavene, den visuelle tilbakemeldingen muliggjort av turtle-biblioteket i Python, og samarbeidsmetoden parprogrammering. Resultatene tyder også på at flere faktorer i den didaktiske situasjonen var forhindrende for elevenes progresjon mot målkunnskapen. Blant de forhindrende faktorene var usystematisk feilsøking av egne programmer blant elevene og misoppfatninger om for-løkker og lokale definisjonsområdet av parametere i subrutiner.

Denne studien bidrar til innsikt i innføringen av programmering i faget Matematikk 1T. Denne studien har identifisert muliggjørende og forhindrende faktorer i den didaktiske situasjonen for elevenes muligheter til å tilegne seg målkunnskapen. Resultatene kan brukes til å videreutvikle og forbedre den didaktiske situasjonen ytterligere før den realiseres i nye klasserom. Videre har denne studien vist at et forskningsdesign inspirert av DE kombinert med TDS kan gi verdifull innsikt i feltet som undersøker krysningen av programmerings- og matematikkundervisning. Følgelig kan tilnærmingen i denne studien inspirere fremtidige forskningsprosjekter innenfor dette feltet.

Preface

This master's thesis marks five years of studies in Natural Science with Teacher Education at NTNU. Programming in the mathematics subject has been an academic passion for me ever since the first long practical training period in the fifth semester. I appreciate the opportunity that I have gotten to spend an entire semester immersing myself in this topic. On that occasion, I want to thank those who have contributed to this report seeing the light of day.

I want to extend a thank you to my supervisor Majid Rouhani for the interesting conversations and discussions about computing education, helpful guidance, and motivational meetings. I want to express my gratitude to the teachers and students who participated in this project.

It has been an inspiring and educational semester. I am proud of what I have accomplished, and I hope this report will be of some use to mathematics teachers. At times it has also been more challenging than I could have imagined. I want to thank my family, partner, and good friends for the unconditional support you give me.

Last but not least, a special thanks to you, Viggo. For all the breakfasts and long days together in our reading room. For the crisis aversion conversations and for sharing the eureka moments together.

Trondheim, June 2022
Anne Margrethe Vestgøte Bosch

Table of Contents

1 Introduction

1.1 Motivation

In recent years, computing has been included in compulsory education in several European countries such as Norway, Sweden, Finland, Estonia, and England (Sanne et al., 2016; Sevik, 2016). A common rationale behind including computing in schools is to equip students with the skills and knowledge needed in the 21st century. Skills such as creativity, communication, and technology production with programming are crucial to the future workforce (Sanne et al., 2016; Sevik, 2016). Computing in compulsory education reaches *everyone*, including the students that will not work with producing digital technology in the future. Computing for everyone has the potential to strengthen computational literacy in the population. Providing access to operate a computationally powerful medium opens up unprecedented possibilities for expression in all fields of society in the future (Guzdial, 2015).

In Norway, computing has been included in primary and secondary education and training as part of the national curriculum renewal in 2020. The approach chosen in Norway has been to incorporate computational thinking and programming in mathematics and other existing school subjects (Bocconi et al., 2018; Ministry of Education and Research, 2017). In addition to the overall goals of introducing computing in compulsory education, it is argued that programming opens up new possibilities particular to the mathematics subject. Programming is a computationally powerful medium for expression that can provide new representations, transforming the students' understanding of the mathematical content (Guzdial, 2015). Other expected benefits are increased motivation, persistence, and performance in mathematics (Forsström & Kaufmann, 2018).

There is, however, limited research documenting the effects (Forsström & Kaufmann, 2018). There are also challenges related to including programming in existing mathematics subjects. The addition of programming can result in content overload. There is also a need for continuing education in programming to support the mathematics teachers operationalizing the inclusion. These challenges can potentially counter the desired benefits (Sanne et al., 2016).

Exploiting synergies between the content in mathematics and programming is a possible approach to counter the challenge of content overload (Guzdial, 2015). However, there is also limited research investigating the effects of cross-curricular approaches where learning objectives in computing and another subject, like mathematics, are both in focus (Waite, 2017).

In summary, new possibilities and challenges follow the inclusion of programming in existing mathematics subjects in Norway. There is a lack of documentation of the effects both internationally and nationally, calling for more research. This study will contribute to the field by addressing the identified gap in the research.

1.2 Problem Statement and Research Questions

This study aims to gain insight into and provide knowledge about how a sequence of teaching situations can be designed with the intent to enable knowledge development in both mathematics and programming. The study is limited to the subject Mathematics 1T (Norwegian Directorate for Education and Training, 2020b) on the upper secondary level in Norway. The specialization project leading up to this master's thesis identified trigonometry as the most pertinent mathematical content knowledge in dual mathematics and programming lessons created by 26 in-service Mathematics 1T teachers enrolled in continuing education (Bosch, 2021a). Therefore, trigonometry was deemed an appropriate branch of mathematics to combine with programming in this study.

The problem statement and research questions addressed in this study are consequently:

> How can a designed didactical situation[1] in Mathematics 1T affect students' opportunities for obtaining the target knowledge in programming and trigonometry?
>
> **RQ1:** Which factors in the didactical situation contribute toward students' possibilities for obtaining the target knowledge in trigonometry?
> **RQ2:** Which factors in the didactical situation contribute toward students' possibilities for obtaining the target knowledge in programming?
> **RQ3:** Which factors in the didactical situation hinder the students' possibilities for obtaining the target knowledge in trigonometry?
> **RQ4:** Which factors in the didactical situation hinder the students' possibilities for obtaining the target knowledge in programming?

Where the specific target knowledge in programming and trigonometry intended in this study are, respectively:

> The students should be able to plan and implement subroutines in problem-solving with programming.
>
> The students should be able to evaluate which trigonometric theorems are appropriate in problems centered around solving triangles and apply the theorems to solve the problems with programming.

1.3 Methodology

The study takes an approach close to classroom practice drawing from the French didactics tradition (Barquero & Bosch, 2015) to answer the research questions. The research methodology applied in the project was inspired by Didactical Engineering (Artigue, 2015), structured in four phases: 1) preliminary analysis, 2) design and a priori analysis, 3) realization and data collection, 4) and a posteriori analysis and validation of the design. The Theory of Didactical Situations (TDS) (Brousseau, 2002) provided a well-researched framework of concepts for lecture structure and analysis, with the target knowledge at the center.

Accordingly, this research project included conducting preliminary analyses of the target knowledge at stake, designing a sequence of lectures, piloting and realizing the designed sequence of lectures, and evaluating factors in the design contributing to or constraining the students' opportunities for obtaining the target knowledge intended in the design.

[1] The concept *didactical situation* is described in Chapter 2

1.4 Structure of the Report

The structure of this report is highly influenced by the four phases in DE conducted in this project. The following paragraph describes the report's structure and highlights the connection to the phases in DE.

The conceptual framework in this study is the Theory of Didactical Situations in Mathematics (TDS). Relevant terms from TDS are presented in Chapter 2. The design research methodology inspired by Didactical Engineering is described in Chapter 3, and the four related phases are described in detail. Methods for data collection and analysis, the pilot intervention, ethical considerations, and discussion of the study's validity are also described in Chapter 3. The preliminary analysis of the content knowledge in trigonometry is presented in Chapter 4. Subsequently, the preliminary analysis of programming subroutines is included in Chapter 5. These two preliminary analyses address the inherent challenges within the target knowledge, educational research related to teaching the target knowledge, and school practices and decisions affecting the teaching of the target knowledge. The preliminary analyses constitute the foundations for the choices made in the lecture design and the associated a priori analysis presented in Chapter 6. The results from realizing the lecture design, called the a posteriori analysis, are presented in Chapter 7. Similarities and differences between the a priori and a posteriori analyses are discussed in Chapter 8. Lastly, the conclusion answers the research questions in Chapter 9.

2 The Theory of Didactical Situations in Mathematics

This chapter presents the conceptual framework used in this study, the Theory of Didactical Situations in Mathematics (Brousseau, 2002), referred to hereafter as TDS. First, the origins and development of the framework are briefly described in Chapter 2.1. Then, a justification of the choice of framework is presented in Chapter 2.2. Lastly, terms and concepts from the framework used in this report are explained in Chapter 2.3.

This chapter aims to give the reader the necessary understanding of terms and concepts from TDS. However, it is worth emphasizing that this conceptual framework is exceptionally complex, and essential ideas such as didactical phenomena are left out as they are not discussed in this study. A translated version of the original presentation of the theory by Brousseau (2002) is recommended for further reading for the curious reader.

2.1 Origins

TDS was founded by the French mathematics education researcher Guy Brousseau. TDS has been developed in the French community of mathematics education since the 1970s (Brousseau, 2002). Brousseau led the TDS research group, and many researchers contributed from 1972 to 1999, including Marie-Jeanne Perrin-Glorian and Claire Margolinas. During this period, *COREM - Center for Observation and Research in Mathematics Education* was significant in the development of the theory (Brousseau, 2002; Strømskag, 2020). The researchers conducted experiments on teachers and students in a natural classroom setting, founding TDS in practice (Strømskag, 2020).

Bridging teaching in practice and rigorous research on teaching was a motivation behind TDS, and it is one of the strengths it holds (Barquero & Bosch, 2015). Since the beginning, TDS has evolved intertwined with the DE research methodology (Barquero & Bosch, 2015). The rich conceptual framework provided by TDS and the research methodology provided by DE combines into a holistic theory and research methodology centered around the target knowledge (Barquero & Bosch, 2015). True to TDS, the research methodology in this study was inspired by DE and is presented in Chapter 3.1.

2.2 Choice of Framework

TDS has been applied in research on all levels of mathematics education from kindergarten throughout university (Barquero & Bosch, 2015), including research on Norwegian mathematics education (Strømskag, 2020, p. 26). Although TDS was initially created for traditional mathematics education, the framework has in recent years been applied in research projects intersecting mathematics teaching and computing (Abdüsselam et al., 2022; Daher et al., 2022; Durand-Guerrier et al., 2019; Gueudet et al., 2014). The rich set of concepts and vocabulary in TDS provides a valuable toolkit for analyzing and discussing factors in the design and realization of the teaching situation (Barquero & Bosch, 2015). Consequently, TDS makes a good foundation for answering the research questions concerned with the factors in the designed teaching situation.

2.3 Concepts

2.3.1 Fundamental Situation

The concept of a *fundamental situation* is connected to the concept of an *adidactical situation*. In the context of a school lesson, an adidactical situation is a situation where a group of students in a class solves a problem using only the feedback provided by the *milieu* they interact with. Central to an adidactical situation is that the teacher stays in the background and does not actively teach. The student groups work independently with the problem-solving.

An adidactical situation can be *fundamental situation* situation for a specific target knowledge (in mathematics), meaning that applying the target knowledge at stake is strictly required to solve the given problem. Designing a fundamental situation for the target knowledge at stake is at the core of TDS and DE. The underlying idea behind fundamental situations is for students to experience the usefulness of the target knowledge in an authentic situation. The belief in TDS is that by independently being able to apply the target knowledge, the students have obtained the target knowledge. Furthermore, by experiencing the usefulness of the target knowledge in an authentic situation without instruction to do so from a teacher, the students can apply the target knowledge in other situations in the future outside of a school setting (Brousseau, 2002; Strømskag, 2020).

2.3.2 Didactical Situation

TDS provides a structure for the teaching situation that is in line with the idea of designing a fundamental situation. The structure involves didactical phases where the teacher actively teaches the students and adidactical phases where students groups work independently on solving a problem.

The proposed structure, the milieu, the participants, and the target knowledge make up a *didactical situation* for the target knowledge. A didactical situation is commonly referred to as a system (Strømskag, 2020). The system view of a didactical situation highlights that all the components of a didactical situation interact with each other. The following three paragraphs describe the structure of a didactical situation and the interactions between the components.

Devolution is a didactical phase and is the start of the didactical situation. The purpose of devolution is to enable the students to work self-reliant on the problem in the following adidactical phase (Brousseau, 2002). The teacher explains the assignment, criteria for success, contents of the material milieu, and rules for the interaction. The students' related questions are answered. Practicalities are handled, such as organizing the class in pairs and handing out material. Central to the devolution is inspiring the students. The students ideally desire to embark on problem-solving as the devolution is over and the adidactical work phase begins (Strømskag, 2020).

Traditionally, the adidactical work phase is divided into three phases: action, formulation, and validation, classifying the mathematical rigor of the students' actions (Brousseau, 2002). These phases are not described here as this study has only incorporated the overall idea and structure of an adidactical work phase. In the adidactical work phase, the students work independently on the problem at hand. They interact with the provided material milieu and their intellectual milieu to solve the problem. As described previously, the adidactical situation is ideally a fundamental situation for the target knowledge. In practice, the students never perfectly follow the intended progression in the adidactical work phase. Therefore, adaptions and improvisation become necessary, and the teacher has to intervene in the students' work. This additional didactical phase is called regulation (Strømskag, 2020).

The didactical situation ends with a didactical phase called institutionalization. In this phase, the teacher decontextualizes the target knowledge from the adidactical situation where it emerged previously (Brousseau, 2002). In this phase, the teacher describes for the first time in the didactical situation what the intention behind the lecture has been. The teacher describes to the students how the target knowledge is helpful in the future. The aim of the institutionalization phase is for students to understand the possible applications and conventional notation and language connected to the target knowledge (Strømskag, 2020).

2.3.3 Milieu

The students interact with the milieu in the adidactical phases of the didactical situation. The milieu consists of both a material and an intellectual part. What is considered part of the milieu is limited to the classroom elements relevant for interaction with the target knowledge. The students physically interact with the material milieu (Strømskag, 2020). The material milieu in this study includes their pair programming partner, a computer with internet access and a programming environment, notes left on the whiteboard, and several printed materials handed out

to the students. The intellectual part of the milieu includes prerequisites held by the students and rules for the problem to be solved. The students must interact with their existing knowledge to mobilize it in the adidactical situation (Strømskag, 2020).

The milieu acts as an opponent for the students in the adidactical phase. A well-designed milieu gives objective feedback whenever the students perform an action. Hence, a functioning milieu helps students evaluate their problem-solving without asking the teacher. Specifically, the feedback from the milieu is central to the fundamental situation. Applying the target knowledge is the optimal solution for the problem in the fundamental situation. The milieu's purpose is to ensure that correct actions give positive feedback, and incorrect actions offer feedback on how to improve the solution (Strømskag, 2020).

2.3.4 Didactical Variables

The designer of a didactical situation has to make design choices. The didactical variables are the design choices the designer is in control of in the didactical situation (Artigue, 2015). There is a range of didactical variables on micro-, meso-and macro levels in a didactical situation. Common for the variables is that the designer is aware of them and makes decisions about every one of them. Using TDS with DE, the preliminary analysis is a basis for the choices.

A didactical variable on the macro level in this study is related to the structure of the teaching sequence. The number of minutes allocated for preparatory work versus the time allocated for the didactical situation is an active choice. A meso-level didactical variable is the choice of the programming environment. Micro-level didactical variables are the choice of student pairings within the class and numbers used in worked examples. Some didactical variables are considered to influence the didactical situation more than others.

2.4 TDS in this Study

It has been an intention in this study to design the lecture sequence to be a fundamental situation for the target knowledge at stake. Consequently, the lectures have been structured in line with the three phases of devolution, adidactical work phases, and institutionalization. The references in this report to *the didactical situation* refer to the sequence of lectures as a whole because the lessons together are part of the system created with the purpose of students obtaining the target knowledge.

The design and a priori analysis in Chapter 6 describes the pertinent didactical variables in this study. The didactical variables are the design choices. The contents of the milieu are didactical variables described. The teacher's actions and intentions for each devolution and institutionalization are described. The planned progression in the adidactical work phases is also described. The didactical variables are subject to discussion in the a posteriori analysis and validation of the study. See Chapters 7 and 8.

3 Methodology

3.1 Research Design - Didactical Engineering

The research design in this study is flexible, qualitative, and inspired by DE. DE is a design research methodological approach originating in mathematics education research in France in the 1980s. It was developed alongside TDS. The methodology emerged due to researchers' growing demand for treating mathematics education as a unique field of research with its characteristic practices separate from pure mathematics and psychology. DE has since spread out from its origin in mathematics education and has been applied in teaching other disciplines, such as physics and sports (Artigue, 2015).

The center of attention in DE is the selected target knowledge, and the design is built around providing optimal situations for students to learn the target knowledge. DE is structured in several phases. First, the researcher carries out preliminary analyses of the target knowledge. The analyses inform the design of a teaching situation. Specifically, the desired outcomes of future realizations of the design are articulated in hypotheses in an a priori analysis. Then the situation is tested in a classroom setting, and data is collected. Lastly, an a posteriori analysis and validation are conducted based on the data collected where the researcher investigates what happened in contrast to the a priori analysis (Artigue, 2015). Didactic engineering was deemed an appropriate choice in this study due to the focus on the target knowledge and the closeness to the field of practice. The implementation of the phases will be described in detail in the following paragraphs.

3.1.1 Preliminary Analysis

The preliminary analyses consist of an epistemological analysis, a didactical analysis, and an institutional analysis. Different aspects of the target knowledge at stake were analyzed in each of the three.

In the preliminary analysis phase, three different dimensions of the target knowledge are under investigation. As the name suggests, aspects regarding the origins of the target knowledge are highlighted in the epistemological analysis. The researcher takes a step back from the problems articulated by the teachers in schools. Concerning this study, a teacher problem can be: "How can programming best be taught in my mathematics classroom?" In the epistemological analysis, the researcher explores the teachers' problems by questioning the history and development of the target knowledge. The researcher looks for inherent obstacles such as conflicting ideas, as the conflicting ideas need to be carefully dealt with in the lecture design. Furthermore, the researcher looks for fundamental situations where the target knowledge was initially developed to solve a real problem. Historical fundamental situations for the target knowledge are crucial in informing the design, as the designed teaching situation should preserve the need to apply the target knowledge (Artigue, 2015; Barquero & Bosch, 2015).

Another dimension of the preliminary analysis is the didactical analysis. The didactical analysis aims to study research about teaching the target knowledge (Artigue, 2015). This analysis is analogous to state-of-the-art knowledge and related work about teaching practices for the target knowledge.

The third dimension in the preliminary analysis is the institutional analysis, where the context for teaching the target knowledge is the subject of the investigations. The institutional analysis examines the teaching context in different layers. The classroom, the school, and the educational system are all relevant to analyze as they affect the scope of possibilities and construct norms for teaching the target knowledge. Becoming aware of norms and established practices leaves room for questioning them and looking for more suitable approaches. Also, the institutional analysis highlights provisions inherent in the institutions that cannot be changed, such as the organization of exams. The results from the pilot and specialization projects preceding this master's thesis informed the institutional analysis.

In this study, the intended target knowledge is dual between programming and trigonometry.

Consequently, two separate preliminary analyses are appropriate. Concerning trigonometry, I have chosen to primarily lean on the preliminary analysis conducted by Stadsvoll (2020) in his mathematics education master's thesis. His analyses were evaluated representative, as they were carried out to design a teaching situation in trigonometry in Mathematics 1T. Chapter 4 summarizes and extends the preliminary analysis of trigonometry conducted by Stadsvoll (2020). Chapter 5 reports on the preliminary analysis of programming conducted in this study.

It is important to note that the preliminary analyses develop and deepen the researcher's understanding of the target knowledge at stake (Artigue, 2015). Hence, the target knowledge and research questions develop accordingly and are subject to change during this phase. In this study, a significant shift along the way was changing the wording used to describe the target knowledge from "[...] use functions [...]" to "[...] apply subroutines [...]". The research design is flexible enough to accommodate the changes, but the changes imply a need for refining and partly redoing the preliminary analyses. This research project had a time constraint of one semester, which limited the opportunity to conduct a complete reiteration of the preliminary analysis.

3.1.2 Design and *a priori* Analysis

Designing a sequence of lectures was the next phase in the project. The majority of the design choices were based on the preliminary analyses. In the associated a priori analysis, the researcher explicitly articulates the hypotheses baked into the design and grounds the choices made. The design and a priori phase determine how to operationalize the findings from the preliminary analyses by creating the design. Important to note is that the generic and epistemic student is at the center of the design. Every individual student has a set of unique prerequisites and experiences in their lives that can not be considered in the design. The design is created with the hypothetical student in mind, considering only the mathematics and programming-related prerequisites. The design phase is at the heart of didactical engineering. The designed teaching situation is the artifact to be realized and validated later by comparing a priori and a posteriori analyses (Artigue, 2015; Barquero & Bosch, 2015). Chapter 6 reports on the design and a priori analysis phase.

3.1.3 Realization and Data Collection

The teaching situation comes to life in implementing the design in the classroom with students. First, the teacher hands over the problem to the students through the devolution phase. Then, it is the adidactical work phase. After the adidactical phase, the teacher institutionalizes the session to decontextualize the target knowledge from the situation.

TDS has an additional regulation phase to handle deviations from the plan in the adidactical phases. The teacher has to act if the students ask for help, get stuck, or deviate significantly from the intended course of action during the adidactical phases. These teacher interventions are called *regulations*. The teacher can repeat the rules and the permitted aids in the situation. For example, she can remind the students that Google is a permitted aid. If a reference to the rules for interaction is not suitable to help the students back into the self-reliant problem solving, she can give them additional information. For example, an additional condition can be added to the assignment steering their focus back to the target knowledge (Strømskag, 2020).

The realization phase includes collecting data to be analyzed in the a posteriori analysis. The data is the basis for the researcher's understanding of students' interaction with the milieu, both the physical and the intellectual. Hence, collecting appropriate data is crucial. Chapter 3.4 describes the data collected during the realization.

3.1.4 *A posteriori* Analysis and Validation

The a posteriori analysis and validation phase evaluates the results from the realization phase against the hypotheses formulated a priori. The intention is to identify divergence and convergence. The results from the a posteriori analysis are included in Chapter 7. Furthermore, this phase aims

to identify and understand the underlying reasons behind the divergence and convergence. Chapter 8 includes a discussion of interpretations of the results. The a priori analysis is concerned with generic and epistemic students, whereas the a posteriori analysis and validation are concerned with the data material generated through realization with real students. Therefore, it is never a complete overlap between the a priori and a posteriori analyses.

3.2 Pilot Realization

The didactical situation was piloted prior to the classroom realizations. The twelve pre-service mathematics teachers who participated in the pilot studied Natural Science with Teacher Education and were recruited through my professional network. There were several intentions behind the pilot realization. The main objective was to test the feasibility of the planned data collection equipment. The positioning of the equipment, such as the camera, is crucial to make the data collection less disruptive and still successfully collect the necessary data (Robson & McCartan, 2016). The pilot realization gave valuable insight into practical problems such as noise recorded from other groups than the one recorded, the time required to rig the equipment up and down, and the time and support required for the participants to submit their programs.

A parallel intention was to test the assignments given to the students and evaluate whether the time frame and material provided were suitable. The pilot participants currently studied university-level mathematics and had one semester of experience in programming from the university. Due to the difference in mathematical and programming prerequisites, it was expected that the level obtained by the pre-service teachers would exceed what could be expected in the actual realizations. The pilot was shortened from $5 \cdot 45$ minutes to $3 \cdot 45$ minutes minutes to accommodate the higher competence level partially. The results and the feedback from the pilot gave valuable insight that led to a downward adjustment of the expectations in the design and a revision of the debugging poster.

3.3 Participants

The classroom realizations were carried out with two regular Mathematics 1T classes in the same upper secondary school. The school is located in a medium-sized city in Norway. The students in the classes were 16 and 17 years old at the time of the realizations. The classes were not selected based on particular characteristics. The teachers responsible for the classes were recruited through my professional network. In class 1, 20 of the 25 students participated in the research project. Three girls and 17 boys participated in class 1. In class 2, all of the 21 students in the class participated in the research project. In class 2, there were eight girls and 13 boys.

3.4 Data Collection

Three methods of data collection were applied in this project. The primary source of data collected was the student programs submitted at the end of sessions 2 and 3 in the classroom realizations. During the adidactical work phases in sessions 2 and 3, one student pair was observed in each of the two classes. Data was not collected during session 1 in each of the classroom realizations as the students had not yet decided on participating in the research or signed the form of consent. Student interviews were conducted after completion of each of the two classroom realizations. Two students from each of the two classes participated in the interviews, respectively. The student programs were collected to gain insight into the products produced by the students. The observation was carried out to gain insight into the work processes involved in producing the programs. The student interviews were conducted to gain insight into the students' opinions about and perceptions of the lecture sequence.

3.4.1 Student Programs

At the end of sessions 2 and 3 in the classroom realizations, the students submitted the programs they had produced during the adidactical work phase in the respective sessions. Class 1 normally used the Microsoft Teams platform, and their team was administered by their teacher. Consequently, Microsoft Teams was used for submissions in class 1. Memory sticks were normally used for submitting work in Class 2, and consequently, memory sticks owned by the school were used for submissions of programs in class 2.

3.4.2 Observation

One working pair of students were observed continuously during the adidactical work phases of the lectures using audio recording, video recording, and screen recording. Data was not collected during the devolution phases, the institutionalization phases, or breaks. The students who were recorded volunteered. The audio recordings were the primary source of observation data resulting in transcriptions. The video and screen recordings were supportive means of observational data collected to provide the necessary context for the recorded dialogue and enrich the transcriptions. The audio was recorded using a smartphone placed on the desk in front of the students for optimal audio quality. The app used to record the audio was *Nettskjema-diktafon* (University of Oslo, 2021). The camera was located behind and above the students. The camera recorded which activities the students were engaged in, such as the division of roles, hand gestures, drawing, programming, silent working periods, and dialogue. The positioning out of sight was chosen to make the students less self-aware during the data collection. The camera used was owned by NTNU. The screen recordings were carried out using the browser version of Panopto[2] on a laptop owned by NTNU. NTNU has agreements with the third-party data collection and temporary storage services Nettskjema and Panopto used in the data collection. The use of these services was approved in advance by the Norwegian Centre for Research Data (NSD). See Appendix A.

The observation was carried out in both classroom realizations. In class 1, only the camera was used to record audio and video, as the camera audio had been sufficient in the pilot realization. However, due to severe noise pollution from other groups, the audio recorded by the camera was too poor to be used for transcriptions. The smartphone audio recording was introduced in the realizations with class 2 to produce usable audio quality. Relatedly, all the screen recordings failed in the observation of class 1 for various reasons, such as closing the laptop or plugging in the charger. The same mistakes were not repeated in class 2. Due to the incomplete audio and screen recordings from class 1, only observational data from class 2 were transcribed and analyzed in this study.

3.4.3 Semi-Structured Interviews

Student interviews were conducted after the classroom realizations to provide deeper insight into the participant perspective of the lecture. The purpose of triangulating the student programs and observations with interviews was to give the students the opportunity to share their opinions, feelings, and perceptions about the lecture sequence and gain insight into the students' understanding of the programs they had written. An interview guide was prepared in advance and is included in Appendix B. The interviews were semi-structured, meaning that questions asked varied slightly from the interview guide (Robson & McCartan, 2016). The order of the questions asked was adapted to fit the flow of the conversation, some questions were omitted if they appeared redundant, and follow-up questions were added to elaborate on answers. The programs written by the students interviewed were available as supporting material to recall their work in each session.

Two interviews were conducted with students from class 1 and one with students from class 2. The first interview was conducted later on the same day as the last session with class 1. As only one of the students showed up, this interview was cut short and discarded because the conversation did not flow as naturally without the other student. Therefore, a second interview with another

[2]https://www.panopto.com/

pair of students from class 1 was arranged one week later. The third interview was conducted with two students from different pairs in class 2. The duration of both interviews was approximately 25 minutes.

Several measures were made to accommodate student welfare in the interviews. Firstly, the students volunteered to participate. Secondly, the interviews were carried out in pairs to provide the students with more time to think and for the students to have a supportive partner. The subject for investigation in the interviews was not the students but the design of the lecture sequence. The focus on improving the design was emphasized at the beginning of the interviews to ensure that the students knew they were not being evaluated. The students were encouraged to share their honest opinions, both positive and negative.

3.5 Data Analysis

The data analysis conducted in this study can be divided into four phases, adapted from the constant comparative method (CCM) (Robson & McCartan, 2016). CCM was chosen because it is a structured process for analyzing data inductively. The first phase happened already in the field during and between classroom realizations and interviews. A notebook was used to record interim analyses and initial interpretations. After both classroom realizations were carried out, the second phase was data processing. The interviews and recordings from the classroom observations were transcribed. The total length of all transcriptions was 49 A4 pages. The screen and video recordings were synced, and actions recorded on the screen were included to enrich the transcriptions. The data processing further contributed to the interim analysis, as notes about interesting situations and statements were recorded in the research notebook. Open coding of the student programs and the transcribed observations and interviews was the third phase. In open coding, the researcher reads through the data material, looking for meaning in each segment, inductively giving code names to the segments that represent their meaning (Postholm & Jacobsen, 2020). This process was carried out supported by the NVivo software, identifying 113 codes. The fourth phase was axial coding, where the codes identified through the open coding were linked together in categories related to the research questions. All of the initial codes were not included in the categories, as some of the codes were not relevant to the categories. The results from the data analysis are presented in Chapter 7.

3.6 Validity and Reliability

There are several aspects of the methodology in this study worth discussing in regard to validity and reliability. The terms validity and reliability have been criticized and attempted to be replaced in qualitative research due to their strong connection to quantitative research (Postholm & Jacobsen, 2020; Robson & McCartan, 2016). However, these terms are commonly used and provide a discourse to discuss and assess the quality of research (Postholm & Jacobsen, 2020; Tjora, 2021). Therefore, these terms will be defined and used in this discussion. *Reliability* can in qualitative research be defined as the researcher's ability to reflect on their own influence on the study and their ability to make the influence transparent to the reader in the report of the study (Postholm & Jacobsen, 2020). The *validity* of qualitative research is related to whether the results from the research actually answer the questions asked (Tjora, 2021).

The quality of flexible and qualitative research depends to a great extent on the quality of the researcher. Experience with qualitative research and the methodology is required to become a good researcher (Robson & McCartan, 2016). This study is the first time I have done research inspired by the DE methodology. It was planned that a co-supervisor familiar with the DE methodology would take part in the project. Unfortunately, she has to withdraw from the project. Consequently, this project has demanded a great deal of independence of me in decision-making related to the methodology and while carrying out the project. The lack of prior experience and methodological supervision can have affected the quality of decisions and execution of this study.

The dual researcher-teacher role can be considered both a strength and a weakness of this study. I

have participated in all phases of the research, including teaching in the classroom realizations. The dual researcher-teacher role was a strength to the realization because I was fully informed about the preliminary analyses and the related design choices. This enabled classroom realizations close to the plan, and therefore it was deemed the better choice. On the contrary, I was a new person introduced to the students. Introducing a new person to a situation affects the situation being observed, posing a threat to the validity of the results (Robson & McCartan, 2016). Habitation is a method used to minimize the effect of introducing a new person to the situation where the research subjects become used to the presence of the observer (Robson & McCartan, 2016). In this study, I attended one double lesson the week before taking over class 2. I introduced myself and the research project, conversed with several students, and acted as an assistant teacher during the lesson. The same habitation process was not feasible with class 1.

As described above in Chapter 3.4, the observational data from class 1 was so incomplete that it was not transcribed or analyzed in this study. Consequently, the data material reporting on observations is sparse. Only observations of one student pair from class 2 have been analyzed. The sparse data material on observations is a threat to the validity of the results, as the results may not be representative for the other student pairs than the pair observed. Observational data from more than one pair, or another pair, could have given other results than those presented in this study. However, the automatic recording of audio, video, and the screen has increased the completeness and quality of the data reporting on the one student pair observed.

3.7 Ethical Considerations

In all research, the researcher needs to consider their ethical responsibility for the participants first, then the investigation, and lastly also themselves as researchers. The order of responsibilities becomes especially relevant when the researcher encounters dilemmas (Postholm & Jacobsen, 2020). The ethical considerations need to be evaluated throughout the study (Robson & McCartan, 2016). The guidelines of NESH, *Guidelines for Research Ethics in the Social Sciences, Humanities, Law and Theology* (NESH, 2019) have been followed in this study to ensure that the research has been carried out in line with the expected ethical standards. This study has included young persons as participants. It has been crucial ensuring that their dignity, integrity, safety, and well-being have been respected throughout the research (NESH, 2019).

3.7.1 Privacy

Protecting the participants' privacy has been a pertinent ethical consideration in this study. Students in the age range 16 to 17 years old participated in this research project. Consequently, the project was subject to notification to the Norwegian Centre for Research Data (NSD). The application was approved with project number 826023. The approval from NSD is included in Appendix A. Several precautions have been taken to ensure the privacy of participants during data collection, data processing, data storage, and in the report. The data collection gear was borrowed from NTNU and has been stored in a location with restricted access. The third-party software used for data collection and temporary storage has data management agreements with NTNU, as described in Chapter 3.4. The only person that has seen and listened to the original recordings is me. The raw data and transcriptions have been stored separately on appropriate online disks for confidential data approved by NTNU. All participant names have been replaced with pseudonyms that are used in transcriptions and throughout the paper. The list linking pseudonyms and real names only existed on paper and has been stored in a location with restricted access. With these precautions, the privacy of the participants should be ensured in the project. Regarding the pilot realization with adult pre-service teachers, the same measures have been taken to ensure their privacy.

3.7.2 Consent to Participate

The students were informed verbally in advance with an associated Q&A session about the research project and data collection. The lecture being subject to investigation, not the students, was emphasized when informing the students. Accordingly, the students were explicitly informed that they would not be graded during the week of the research project. In addition to the verbal information, the students were given the same information in a printed form of consent. The form of consent is included in Appendix C. They were given time to read through the form at home. The language in the form of consent was adapted to fit the age group. The information provided to the students about the aims of the research project and the data collection strived to provide them with transparent and understandable information, in line with the NESH guidelines (NESH, 2019). This was to ensure the students had adequate knowledge to make an informed decision about participation. Twenty students from class 1 and all 21 students in the class gave their consent to participate in the study. The participants in class 1 that did not consent were not in the classroom during the data collection and got alternative lessons from their teacher. An alternative was provided to ensure the students had a choice not to participate. The students that were recorded during the data collection and the students interviewed volunteered. These measures were made to ensure that all participation in the project was voluntary, in line with NESH (2019).

Regarding the pilot realizations, the same measures were made to ensure informed consent. The prior information about the project was provided through Facebook and forms of consent. The pilot form of consent is included in Appendix D.

3.7.3 The Dual Researcher-Teacher Role

There are several ethical aspects related to the dual Researcher-Teacher role. As a teacher, I was exposed to personal information about the students during the class realizations. For instance, some students shared their interests with me during and between lessons. Relatedly, I was informed in advance about special needs in the class. I have a duty of confidentiality regarding this information, and it has been treated accordingly.

The welfare of the students was also my responsibility while teaching. The ethical principle of valuing student welfare over the production of results for the research has been followed. For instance, this principle became relevant in an interview when one of two students did not show up. The interview was cut short to ensure the welfare of the student missing their partner in the interview.

4 Preliminary Analysis of Trigonometry

This chapter is mainly based on the results of Vebjørn Stadsvoll's (2020) master's thesis. He designed a set of teaching situations to introduce trigonometry in Mathematics 1T. TDS and a methodology close to DE were used in the design, and consequently, a preliminary analysis was produced. This chapter summarizes the relevant findings from the preliminary analysis conducted by Stadsvoll and further extends his preliminary analysis. First, Stadsvoll's findings concerning the trigonometric functions $\sin(x)$ and $\cos(x)$ are presented in Chapter 4.1. Then, in Chapter 4.2, attention is drawn to Stadsvoll's findings concerning degrees and radians, two angle measuring units. In Chapter 4.3, the findings from Stadsvoll's preliminary analysis are applied in three trigonometric theorems to highlight the implications for the didactical situation in this study. The three theorems are the SAS Theorem, The Law of Cosines, and The Law of Sines. In the teaching designed teaching situation, the three theorems are applied by the students.

4.1 Two Conflicting Definitions

Trigonometry studies angles and ratios in 2- and 3- dimensional figures. There are six trigonometric functions; cosecant, cosine, cotangent, secant, sine, and tangent. Two of these are relevant in this study, namely sine and cosine. In mathematical notation, $\sin(x)$ and $\cos(x)$, where x is an angle measured in degrees or radians (Stadsvoll, 2020).

The trigonometric functions are defined in two ways. The first set of definitions is based on rations between sides in a right triangle, further referred to as the triangle definition. Let θ be one of the angles in the triangle. The sides in the right triangle are named opposite, adjacent, and hypotenuse based on their relative position to θ (Stadsvoll, 2020). Figure 1 illustrates a right triangle with reference angle θ and naming conventions for sides. The trigonometric functions are defined accordingly:

$$\sin(\theta) = \frac{\text{opposite}}{\text{hypotenuse}}$$
$$\cos(\theta) = \frac{\text{adjacent}}{\text{hypotenuse}}$$

$$\text{where } \theta \in \langle 0^\circ, 90^\circ \rangle$$

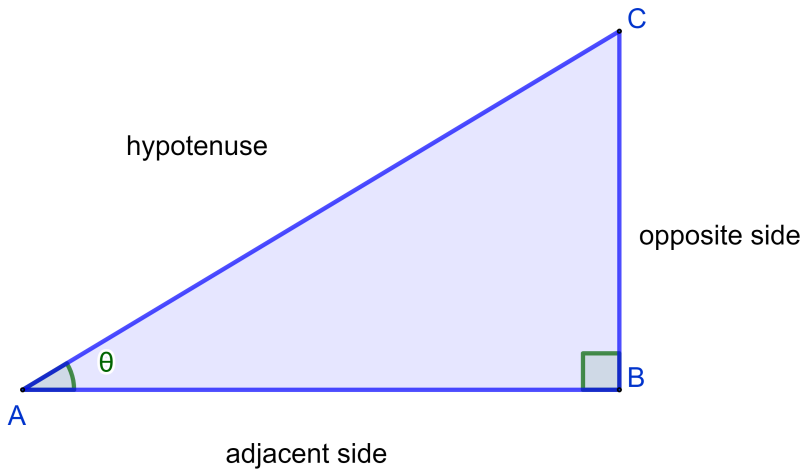


Figure 1: *Illustration of a right triangle with reference angle θ and naming conventions for the sides.* Adapted from "Et design av en didaktisk situasjon for introduksjonen av trigonometri etter prinsippene i didaktisk ingeniørvirksomhet," by V. Stadsvoll, 2020, p. 13 (https://hdl.handle.net/11250/2778352). CC BY 4.0.

The second set of definitions is based on the unit circle, further referred to as the unit circle definition. In the cartesian coordinate system, the unit circle has its center in the origin point $O = (0, 0)$. The radius of the unit circle is 1. The sine and cosine are given by the coordinates of a point on the perimeter of the unit circle. Let angle θ span counter-clockwise from the origin, and let the x-axis be the right leg of θ. Extend the line along the left leg of θ so that the line intersects the unit circle. The intersection is called point P. Then, $\cos(x)$ is the x-coordinate of P, and $\sin(x)$ is the y-coordinate of P. The domain of θ is not restricted in the unit circle definition, unlike the domain in the triangle definition (Stadsvoll, 2020). Figure 2 illustrates the unit circle definition of the trigonometric functions.

Elaborating on the unit circle definition provided by Stadsvoll (2020), the sign of $\sin(x)$ and $\cos(x)$ varies in the four quadrants of the cartesian coordinate system. Focusing on the domain of angles in a triangle, the first and second quadrant is of interest. Here, $\sin(\theta) \in \langle 0, 1\rangle, \quad \forall\theta \in \langle 0°, 180°\rangle$. More importantly, $\sin(x)$ is positive for all angles in triangles. In contrast, $\cos(\theta) \in [0, 1\rangle, \quad \forall\theta \in \langle 0°, 90°]$ and $\cos(\theta) \in \langle -1, 0\rangle, \quad \forall\theta \in \langle 90°, 180°\rangle$. In other words, $\cos(x)$ is positive for acute angles, 0 for right angles, and negative for obtuse angles. $\cos(x)$ is an injective function on the domain of angles in a triangle, and $\sin(x)$ is not (Weisstein, n.d.-c). It can be illustrated through implication:

$$\forall\theta, \alpha \in D, \cos(\theta) = \cos(\alpha) \implies \theta = \alpha$$
$$\forall\theta, \alpha \in D, \sin(\theta) = \sin(\alpha) \not\implies \theta = \alpha,$$
$$\text{where } D = \langle 0°, 180°\rangle$$

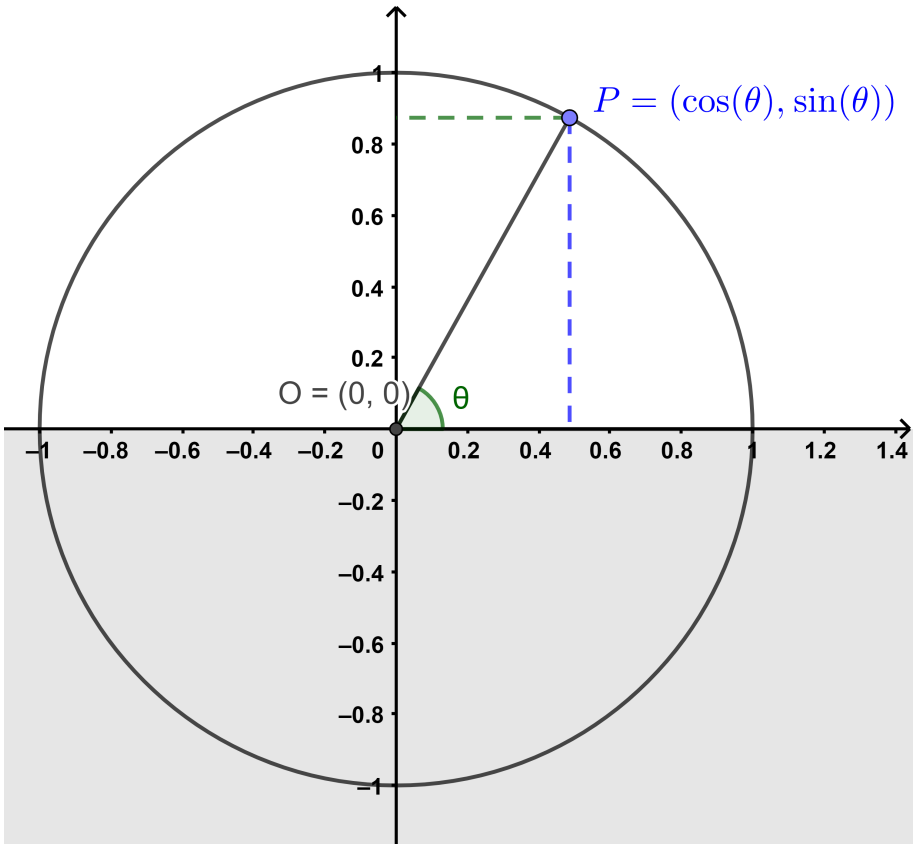


Figure 2: *Illustration of the unit circle definition of* $\sin(x)$ *and* $\cos(x)$. *Note* The shaded area marks the domain of angle theta that lies outside the domain of angles in triangles. Adapted from "Et design av en didaktisk situasjon for introduksjonen av trigonometri etter prinsippene i didaktisk ingeniørvirksomhet," by V. Stadsvoll, 2020, p. 15 (https://hdl.handle.net/11250/2778352). CC BY 4.0.

As Stadsvoll (2020) points out, both definitions of the trigonometric functions have appropriate applications. The triangle definition is suitable when working with right triangles or decomposing

shapes into right triangles, as the associated calculations are easy to perform. The unit circle definition is more general than the triangle definition and can be applied to problems with all possible triangles, not restricted to right triangles.

The two definitions present conflicting ideas, which lays the foundation for an epistemological obstacle regarding the domain of angle θ. Stadsvoll (2020) also emphasizes the choice of definition as a recurring issue in research concerning the introduction of trigonometry. The conflicting definitions of the trigonometric functions $\sin(x)$ and $\cos(x)$ is a challenge that influences the application of The SAS Theorem Area Formula, The Law of Sines, and The Law Of Cosines. Consequently, the challenge needs to be taken into consideration in this study. The impact of the conflict on the didactical situation in this study will be further discussed in chapter 4.3.

4.2 Units for Angle Measurement

An angle can be measured in different units, among them degrees and radians.

The unit of degrees probably comes from the Babylonians. They used it to measure arc lengths, and consequently, the numeral value of a degree depends on the circle radius. The Babylonians used the sexagesimal number system using base 60. Therefore, the Babylonians often gave a circle radius $R = 60$ in trigonometric problems, making the measure of degrees comparable. By approximating π to 3, the circumference of a circle is naturally divided into 360 parts, called degrees (Stadsvoll, 2020). See the equation below:

$$O = 2\pi R \approx 6R = 6 \cdot 60 = 360$$

Radians is a measuring unit that encapsulates the relationship between radius and arc length in a circle. One *radian* is defined as an arc with the length of one circle radius. 2π radians corresponds to an arc the size of a full circle. This relationship is independent of the value of the radius (Stadsvoll, 2020). Figure 3 illustrates the radian angle measure.

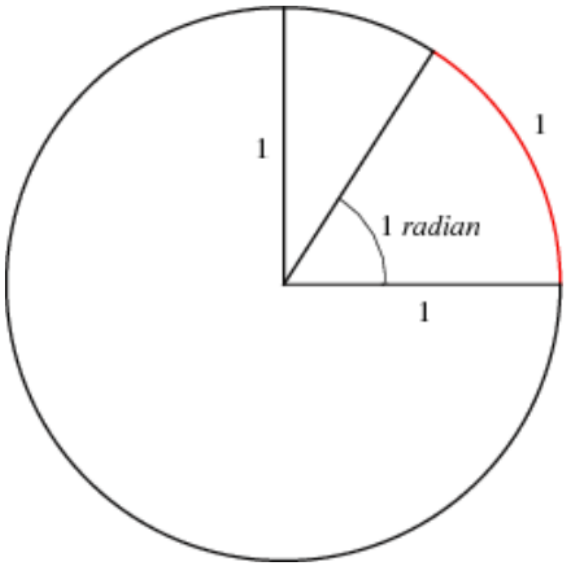


Figure 3: *The radian angle measure illustrated in a unit circle.* From "Et design av en didaktisk situasjon for introduksjonen av trigonometri etter prinsippene i didaktisk ingeniørvirksomhet," by V. Stadsvoll, 2020, p. 22 (https://hdl.handle.net/11250/2778352). CC BY 4.0.

It is common to use the degree as a measuring unit for angles in teaching trigonometry. The concept of radians is difficult for students accustomed to degrees. Especially, interpreting radians as degrees is a common mistake. For example, the students interpret $\sin(30)$ as $\sin(30°)$. The convention in

Norwegian mathematics education is to measure angles in degrees through Mathematics 1T and introduce radians in subsequent mathematics courses (Stadsvoll, 2020). The use of two different units for measuring angles sets the stage for an epistemological obstacle that needs to be handled in the teaching situation in this study.

4.3 Implications

In this chapter, the epistemological challenge addressed in Chapter 4.1 is applied to The SAS Theorem, The Law of Cosines, and The Law of Sines. The three theorems build on the definitions of either $\sin(x)$ or $\cos(x)$. As a consequence, the inherent challenges of $\sin(x)$ and $\cos(x)$ are continued in the theorems. This chapter highlights how the inherent challenges affect the application of the theorems. The content in this chapter reports on the result of my own work extending the preliminary analysis of trigonometry.

4.3.1 The SAS Theorem

Multiple combinations of known side lengths (S) and angle sizes (A) uniquely determine a triangle (Weisstein, n.d.-g). The Side-Angle-Side Triangle Congruence Condition, further referred to as the SAS Theorem, is one of them. The SAS Theorem states that a triangle is uniquely determined by specifying two sides and the angle between them (Venema, 2012; Weisstein, n.d.-f).

As the SAS Theorem uniquely determines a triangle, the area of said triangle is also uniquely determined. The formula for calculating the area using SAS is derived from the triangle definition of $\sin(x)$ and the standard formula for the area. The formula is further referred to as the SAS Theorem Area Formula. A derivation of the SAS Theorem Area Formula inspired from Venema (2012) is included in full in Appendix E. The end product of deriving the SAS Theorem Area Forumula is presented here. The sides c and a, and angle B are known in triangle $\triangle ABC$. Let c be the baseline of $\triangle ABC$ and let h be the height. The area T is given by the following formula:

$$T = \frac{1}{2} \cdot c \cdot a \cdot \sin B$$

The SAS Theorem is valid for all possible triangles in the Euclidian geometry (Venema, 2012). The triangle angle sum of $180°$ restricts the known angle B domain to be $B \in \langle 0°, 180° \rangle$. The SAS Theorem Area Formula has the same domain of angle B. The conflict between the two definitions of $\sin(x)$ is baked into calculating the area with the formula. The triangle definition is used to derive the formula, but the domain of the angle B is not restricted to $B \in \langle 0°, 90° \rangle$, in line with the unit circle definition. The integrated mix of definitions impacts the didactical situation in this study. The SAS Theorem Area Formula is incorporated into the curriculum of Mathematics 1T (Norwegian Directorate for Education and Training, 2020b). It is central to the area calculations implemented in session 3 in the didactical situation, as described in Chapter 6.

4.3.2 The Law of Sines

The Law of Sines is an equivalence of ratios in a triangle. Let a, b, and c be the sides in a triangle. Let the angles opposite be respectively A, B, and C. The Law of Sines then gives the equivalence

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C}$$

(Weisstein, n.d.-e)

Like the SAS Theorem Area Formula, the Law of Sines is valid for angles $\theta \in \langle 0°, 180° \rangle$. The Law of Sines can be directly derived from the triangle and unit circle definitions of $\sin(x)$. The proof has two parts. The first part is centered around heights in an acute triangle, and the second is centered around heights in an obtuse triangle. The height in an obtuse triangle lies outside the

triangle, so the complementary angle is used to prove the theorem. Hence, the unit circle definition of $\sin(x)$ is needed. An acute and an obtuse triangle is provided in Figure 4 for illustration. An associated proof adapted from Venema (2012) is included in full in Appendix F.

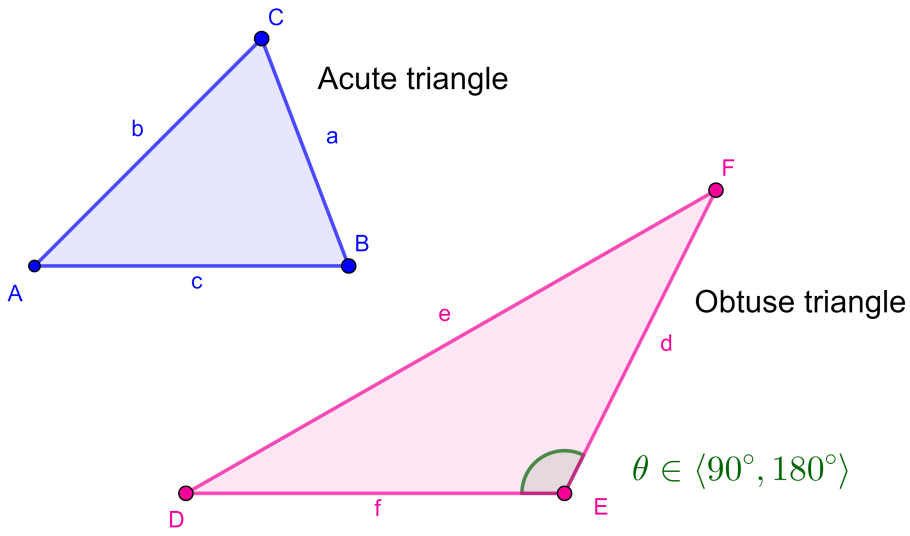


Figure 4: *Illustration of an acute triangle $\triangle ABC$ and an obtuse triangle $\triangle DEF$.*

4.3.3 The Law of Cosines

The Law of Cosines is also referred to as the general Pythagorean theorem, as it can be applied in triangles that are not right (Venema, 2012). Let $\triangle ABC$ be any triangle. Two sides a and b and the angle C between them are given. Then the square of the last side c is,

$$c^2 = a^2 + b^2 - 2ab \cdot \cos C$$

(Weisstein, n.d.-d)

The Law of Cosines can be proved using the Pythagorean Theorem together with the triangle definition of $\cos(x)$ for acute triangles and the unit circle definition of $\cos(x)$ for obtuse triangles. A proof adapted from Venema (2012) is included in Appendix G.

The Law Of Cosine's independence from $\sin(x)$ has a significant impact on solving triangles when applying the results from Stadsvoll (2020). The Law of Cosines can be rewritten to find an angle θ when three sides are given. Angle θ is uniquely determined for all angles possible in triangles, that is, $\theta \in \langle 0°, 180° \rangle$ due to the unit circle definition. For instance, the sides a, b and c are given in triangle $\triangle ABC$. Below, the Law of Cosines is rewritten to determine angle C

$$\cos C = \frac{c^2 - b^2 - a^2}{-2ab}$$

Similarly, angles A and B can be determined using the Law of Cosines.

5 Preliminary Analysis of Programming

This chapter reports on the preliminary analysis conducted in this project of the target knowledge related to programming. The preliminary analysis is the first phase in the DE research methodology and provides the foundation for choices made in the design of the didactical situation. First, the terms *programming* and *subroutines* are defined in Chapter 5.1. Then, the results from the epistemological analysis of subroutines are presented in Chapter 5.2 aiming to identify inherent obstacles. The results from the didactical analysis of programming are presented in Chapter 5.3, analogous to the state-of-the-art of didactical research on teaching programming in K-12 education. Lastly, the results from the institutional analysis are provided in Chapter 5.4, aiming to give insight into the institutional practices influencing the teaching of the programming component of the target knowledge. Programming was the scope of the didactical and institutional analyses rather than just subroutines, as the scope of subroutines was deemed too narrow.

5.1 Definitions

This study is concerned with target knowledge in programming. Consequently, it is appropriate to include the definition of programming in this study. Due to the context of the study, the definition chosen is inspired by the government documents regarding programming in the Norwegian curriculum.

The target knowledge in this study is particularly concerned with the application and implementation of subroutines in programming. As this preliminary analysis has provided insight into the origins of subroutines, it was deemed appropriate to include a definition close to the origins.

5.1.1 Programming

Programming is the process of developing a computer program that determines how a computer or other electronic devices should function while the program is active or running (Rossen, 2019). Programming goes beyond implementing the program in a language that a machine can interpret. The whole process, from identifying and understanding the problem at hand, planning the execution, implementing the program, troubleshooting, and continuously improving the program, is included in the process (Sevik, 2016).

5.1.2 Subroutine

A subroutine is both a conceptual idea and an implemented sequence of instructions. A subroutine is a self-contained part of a computer program. The subroutine is made up of a sequence of instructions and aims to solve a sub-problem of one or more computer programs (Dasgupta, 2014).

5.2 Epistemological Analysis

5.2.1 Origins of Subroutines

The origins of subroutines date back to the 1940s, related to the construction of computers. A central contributor to the invention of subroutines is John Mauchly. The need for subroutines emerged from computer programmers performing a specific set of instructions frequently, but not frequently *enough* to build the instruction set into the computer. Mauchly suggested the notion of subroutines as small programs stored outside the main program. A specific subroutine could be called from the main program whenever needed. Hence, the subroutine would reduce the need for rewriting the same code lines often (Dasgupta, 2014, p. 141).

Bringing to life the subroutine ideas of Mauchly was David Wheeler. He was one of the first constructors of subroutines for the ESDAC machine. He recognized how programming problems were

similar but not identical (Dasgupta, 2014, p. 142). Wheeler formulated problems and solutions related to the implementation of subroutines. To accommodate solving similar problems, he argued that a subroutine needed to be "[...] sufficiently general by using parameters that could be set to specific values for each instance of the subroutines' use [...]" (Dasgupta, 2014, p. 143).

Connected to the generalizability of subroutines, the subroutines needed to be callable from different main programs stored in different locations. Wheeler managed to construct *closed subroutines* and the related *Wheeler jump*. The Wheeler jump describes how the closed subroutine can be called from any location n in the main program during execution. Then, the closed subroutine is executed in sequence. Afterward, the execution jumps back to location $n+1$ in the main program (Dasgupta, 2014, p. 143).

Wheeler furthermore implemented the closed subroutines through a subroutine library for the EDSAC machine. In the implementation, he accommodated the subroutine generalization principle. Arguments could be passed to the subroutines, allowing the subroutines to solve different problem instances. Hence, he is recognized as the first programmer to construct a set of subroutines that were application-independent building blocks ready to use by other computer programmers (Dasgupta, 2014, p. 145).

Summary
The term *subroutine* describes both a conceptual artifact and the associated implementation of an instruction set carrying out a specific action. A subroutine can be called within the execution of another program, and after the subroutine is carried out, the main program execution continues in the right place. Inherent from the origins of subroutines is the ability to solve a family of related problems. The subroutine was meant to be called with different parameter values to solve slightly different problem instances.

5.2.2 Implementation of Subroutines

There are several types of subroutines. They can be distinguished by their number of parameters and return values. A *function* is a subroutine that returns one or more values, whereas a *procedure* does not return any value. Some programming languages distinguish between functions and procedures, whereas other languages do not (Mitchell, 2003, p. 170).

All subroutines are technically implemented as functions in Python. The return value is None unless it is overridden (Pilgrim, 2009, p. 3). A function is declared through a dedicated programming construct and the reserved def keyword. The return keyword is reserved in Python for overriding the default return value. When a function has been declared, it can be called by referencing the function name (Pilgrim, 2009). Python subroutines that do not override the default value will be further referred to as procedures for simplicity due to the difference in behavior from functions with an overriding return value.

5.3 Didactical Analysis

5.3.1 PRIMM

PRIMM is a method for teaching programming to students on the K-12 level (Sentance & Waite, 2017). It is challenging for students to write computer programs before they can read and understand (trace) programs written by themselves and others. The challenge can become visible in the classroom when students stare at a blank screen, not knowing where to start. PRIMM addresses the challenge by offering an approach where students progress in five steps, from tracing programs to writing their own programs. The five steps are Predict-Run-Investigate-Modify-Make, and the name PRIMM is an acronym for the steps (Sentance & Waite, 2017; Sentance et al., 2019a).

In the **predict** stage, the teacher provides students with a program. The students work collaboratively to predict the output of running the program and write down a hypothesis (Sentance et al., 2019b).

The **run** stage follows the predict stage. The students download the program to their own computers and run the program to test their hypothesis. If there is a discrepancy between the expected result and the actual result, it is a symptom that students do not understand the program (Sentance et al., 2019b).

The **investigate** stage, code comprehension is in focus. The students investigate the functioning of a program. The students answer questions about program constructs on different abstraction levels. For example, the teacher can ask about the functioning of a single line or the overall goal of the whole program. Students usually work collaboratively in this stage to develop their programming vocabulary (Sentance et al., 2019b).

In the **modify** stage, students extend or change the functionality of a program. The changes can start small and progress to more fundamental changes, providing opportunities for differentiation within the class. At this stage, the students start developing ownership of the program(Sentance et al., 2019b).

The **make** stage is where students create new programs. Here, the students can be met with a blank canvas and a problem description. In this stage, they build on their existing programming knowledge and borrow snippets from other programs to apply in a new setting.(Sentance et al., 2019b).

PRIMM is a flexible method where teachers can apply the stages they deem necessary in a lesson. The progression through the stages can be implemented through several lessons. The stages can be repeated in iteration, and the sequence of stages can be modified. Implementation is flexible to the class in question and the content being taught. Crucial to PRIMM are the core principles of scaffolding program comprehension before writing programs, gradually enabling students to apply their programming knowledge in their own programs, building up ownership progressively, and working collaboratively (Sentance & Waite, 2017; Sentance et al., 2019a).

5.3.2 Pair Programming

Pair programming is a collaborative programming practice where two programmers work together on one computer. There are two different roles involved in pair programming. The 'driver' writes the program and is in control of the mouse and keyboard on the computer. The 'navigator' is also an active member of the pair by supporting the driver. The navigator's responsibilities include catching bugs in the program, keeping track of the overall plan for the programming session, and retrieving information such as looking up documentation. The navigator can have access to an additional computer for information retrieval, but it should not be used for programming. In pair programming, the driver's computer screen displaying the program is visible to both participants. It is common for the roles to switch between partners within a session and for pairs to switch regularly within a team (Beck & Andres, 2004).

Pair programming originated as one of the core practices of Extreme programming (XP). XP is a framework for professional software development in teams (Beck & Andres, 2004). The founder of XP, Kent Beck, describes his vision of XP:

> XP is my attempt to reconcile humanity and productivity in my own practice of software development and to share that reconciliation. (Beck & Andres, 2004, p. 3)

From its origins in the software development industry, pair programming has been researched in educational settings on the university level (Luxton-Reilly et al., 2018), and recently also in K-12 programming education (Denner et al., 2014). Pair programming is an educational approach to teaching programming that is proven through research to be more effective than solo programming (Luxton-Reilly et al., 2018, p. 70). The aim is to achieve benefits connected to productivity and participant welfare, as described by Beck and Andres (2004).

Focusing the attention on pair programming at the K-12 level, Campe, Green, and Denner have published a Pair programming toolkit for K-12 computing education based on research (Campe et al., 2019). They describe the documented benefits of pair programming on the K-12 level

of education and have created guidelines for implementing pair programming in classrooms. The toolkit also provides classroom materials. Pair programming being *effective* means that it increases the quality of the work (Beck & Andres, 2004). In the K-12 context, increased quality of the work means that the student programs have fewer bugs, the students have a greater increase in programming knowledge, and the persistence is better in problem-solving (Campe et al., 2019). Regarding the welfare aspect, a beneficial effect of pair programming is that students enjoy it (Campe et al., 2019).

The purpose of the toolkit is to support teachers so that the implementation of pair programming in the classroom actually results in the desired benefits. Through the toolkit, it is emphasized that student pairings need to be paid attention to. To support the building of programming skills, pairing students with friends is a good strategy. It is more important than having similar programming knowledge (Campe et al., 2019, p. 9).

The toolkit also provides guidelines for teaching pair programming to the students. The teacher needs to make sure the students understand the roles involved and their common and respective responsibilities. It is beneficial to highlight to the students why pair programming is valuable for them in a language they understand. Examples of value for students are that help is available instantly, they write programs with fewer mistakes, and they practice collaboration skills useful in other school subjects and future work (Campe et al., 2019).

It is not sufficient to teach the methodology once and leave the students to work self-reliantly. The toolkit emphasizes how scaffolding the students in their work is essential for a successful implementation of pair programming in the K-12 classroom. If the student pairs deviate from the principles of pair programming, the teacher should guide the students back to the methodology. Positive feedback on successful pair programming is even more important (Campe et al., 2019).

5.3.3 Debugging

Dealing with errors and troubleshooting is a substantial component of the programming process for all programmers, both novices and experienced software developers (Michaeli & Romeike, 2019a). *Debugging* is a term used for describing the process of finding and fixing errors in computer programs (Michaeli & Romeike, 2019b). Dealing with errors is particularly hard for novices, and it is a source of frustration and helplessness. Students getting 'stuck' in unstructured trial and error attempts to fix errors in their programs is a problem. K-12 teachers report running around from student to student to help the students get unstuck. Therefore, increasing the students' self-reliance in debugging is a productive strategy in teaching programming (Michaeli & Romeike, 2019b).

To support students to become more self-reliant in their debugging, Michaeli and Romeike (2019b) found that teaching a systematic debugging process increased the students' debugging performance. The systematic method for teaching debugging to students is similar to the scientific method. Hypotheses are formulated and tested, leading to new and refined hypotheses being tested (Michaeli & Romeike, 2019b).

The instructor can provide students on the K-12 level with a poster to support their progression in systematic debugging as they work on debugging assignments (Michaeli & Romeike, 2019b). The poster is structured around four cycles of step-wise hypothesis formulation and testing for locating, determining, and fixing the error. See Figure 5. The first cycle includes procedures for handling compile-time errors. The second cycle includes procedures for handling runtime errors. The last cycle includes procedures for handling logical errors. As introducing new bugs is common when novices debug, reverting unsuccessful changes is an emphasized step in each of the cycles. It is debated whether handling compile-time errors can be regarded as debugging or not. Compile-time errors are consciously included in the systematic debugging process described on the poster as they are a significant cause of hurdles in the K-12 context Michaeli and Romeike (2019a).

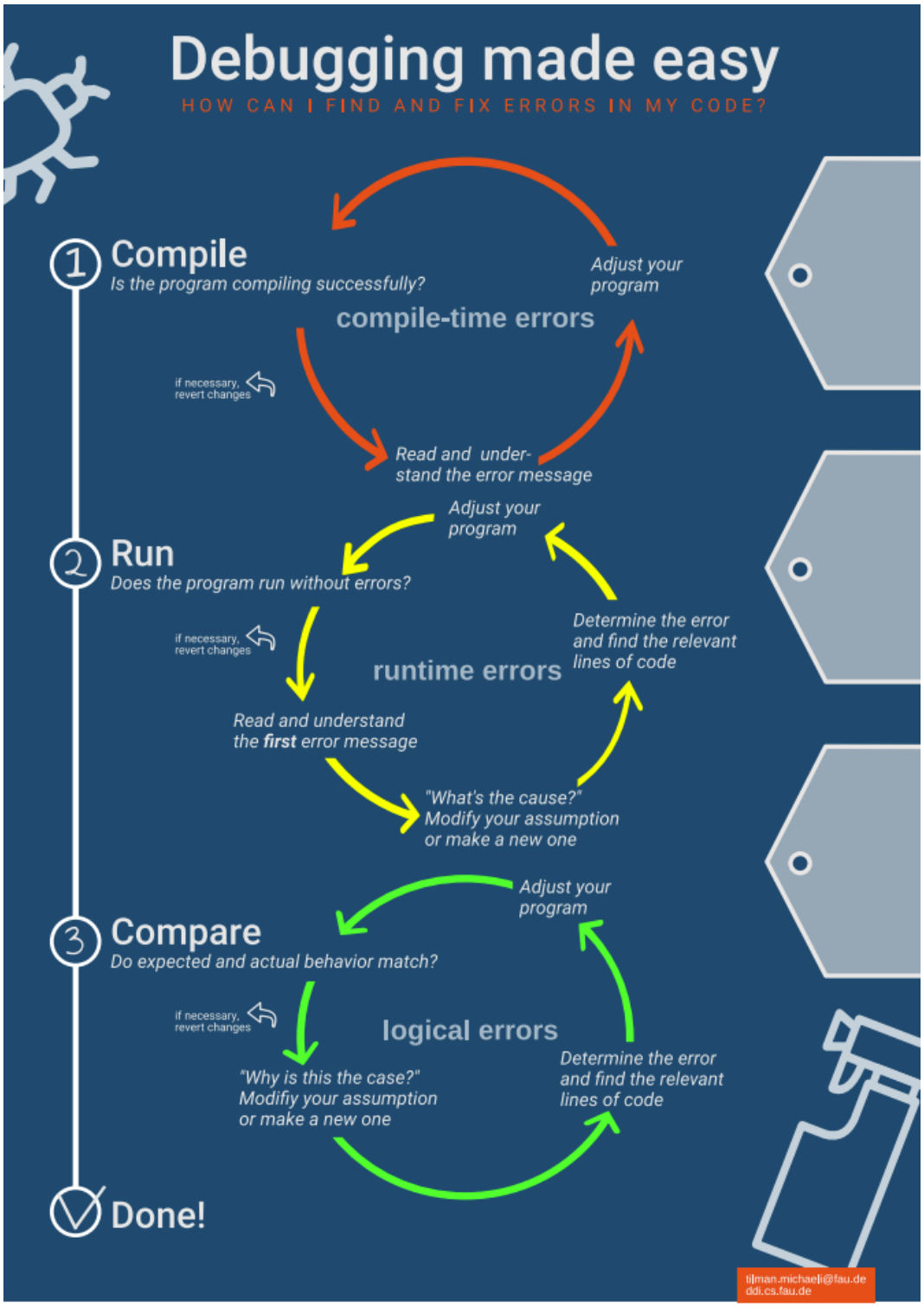


Figure 5: *Poster conveying a systematic debugging process.* From "Improving Debugging Skills in the Classroom: The Effects of Teaching a Systematic Debugging Process," by T. Michaeli and R. Romeike, 2019, *Proceedings of the 14th Workshop in Primary and Secondary Computing Education, 2019*, Article 15 (https://doi.org/10.1145/3361721.3361724). Copyright 2019 by Tilman Michaeli and Ralf Romeike. Reprinted with permission.

A design proposal for future development of the debugging poster is to include concrete debugging strategies for supporting the teaching of self-reliant debugging skills further (Michaeli & Romeike, 2019b). One example of a concrete debugging strategy for locating an error is code tracing, where every program line is read. Code tracing can be done top to bottom or in the sequence of lines

executed in the program (Li et al., 2019). Another concrete debugging strategy for locating the error is program chunking, where the program is divided into smaller parts to isolate the error. Program chunking can be done by commenting out code 'chunks,' isolating a functionality in a subroutine, or other methods (Li et al., 2019).

Another means to support students' self-reliance in debugging is demanding autonomy from students. A literature review on debugging in K-12 classrooms found that teachers who expect self-reliance from their students in error handling reported fewer error-handling problems, especially regarding compile-time errors (Michaeli & Romeike, 2019a). In other words, the teacher's response to students who struggle with debugging can influence the students' debugging performance.

5.4 Institutional Analysis

I have conducted two research projects leading up to this master's thesis. The first research project and the associated report were part of the course *Research Methods in Mathematics and Science Education* weighted 7.5 ECTS (NTNU, n.d.-a). The project is further referred to as the pilot project. The second research project and the associated report were the contents of the course *Computer Science Specialization Project* weighted 15 ECTS (NTNU, n.d.-b). The project is further referred to as the specialization project.

Both projects concerned the same phenomenon investigated in this thesis, namely the implementation of programming in the subject Mathematics 1T. Both projects investigated the phenomenon from teachers' perspectives. The findings from the two previous research projects inform the institutional context in which the target knowledge in this study is taught. Student prerequisites expected by teachers and established practices regarding tools, languages, and environments (TLEs) were extracted from the projects and applied in this study. Chapter 5.4.1 summarizes the findings from the pilot project. Chapter 5.4.2 describes pertinent findings from the specialization project.

The last part of the institutional analysis was to collect relevant information about the classes involved in this study. Chapter 5.4.3 provides an overview of the programming-related student prerequisites and TLEs used in the specific classes investigated in this study.

5.4.1 Results from the Pilot Project

The pilot project was a qualitative research project conducted in 2020. The project investigated the research question: "What resources do three mathematics teachers require to support their work with the implementation of programming in the subject Mathematics 1T in connection with the Subject Renewal?". Three Mathematics 1T teachers were interviewed. The Subject Renewal was operationalized in the fall semester of 2020. Hence, the interviews were conducted during the first year programming was included in Mathematics 1T (Bosch, 2021b). The results from the project were three types of resources requested by the teachers for supporting the implementation of programming into Mathematics 1T.

The first resource was continuing education in programming for teachers. The desired contents of the continuing education varied based on the teaching experience and prior programming experience among teachers (Bosch, 2021b).

The second resource was national guidelines for programming in Mathematics 1T. The national-level expectations for students' programming proficiency are perceived as unclear. The teachers want guidance on the ambitions they should have for their students regarding programming in the specific subject of Mathematics 1T. They especially want such guidelines at the beginning of the introduction, as they have little or no experience even with programming. Clearer expectations of acquired programming skills are necessary for teachers to be able to construct learning objectives and plan lessons (Bosch, 2021b).

The third supporting resource requested is tightly connected to the need for national guidelines. In the Mathematics 1T subject curriculum, students are expected to apply programming as a tool in

problem-solving. Teachers experience not having enough time to teach students to apply programming in Mathematics 1T for mathematical problems. Instead, the time is spent on introductory programming as most students have no programming prerequisites. Consequently, the Mathematics 1T teachers require that their students obtain introductory programming skills in primary and lower-secondary education. If that goal is achieved, they will be able to spend the time applying programming (Bosch, 2021b).

5.4.2 Results from the Specialization Project

The specialization project was carried out during the fall of 2021. This qualitative research project investigated the problem statement: "How do in-service teachers enrolled in programming training envision integrating programming into Mathematics 1T?". Twenty-one lecture plans incorporating programming into Mathematics 1T were analyzed. Twenty-six in-service teachers enrolled in learning programming through continuing education designed the plans (Bosch, 2021a).

One of the main findings from the projects was a connection between expected programming-related student prerequisites and the learning objectives in the lecture plans. Three approaches were identified. The student programming prerequisites varied among the identified approaches. The first approach was based on no prior programming experience among the students, and the learning objectives concerned programming. Lecture plans within the second approach assumed existing programming prerequisites among the students, and the learning objectives were focused on mathematics. The third approach was the most common in the data material, identified in 13 of the 21 lecture plans. Lecture plans in this category combined programming-related and mathematics-related learning objectives (Bosch, 2021a).

Investigations of the lecture plans' programming-related student prerequisites and learning objectives gave insight into the students' programming concepts and skills expected to be obtained by the end of the subject Mathematics 1T. The list abstracted in the specialization project from the lecture plans is presented below.

- Definite loops
- Indefinite loops
- Conditionals
- Variables
- Defining and calling functions
- Textual output (print)
- Primitive data types
- Arithmetic operators
- Import and use of third party libraries
- Recursion
- What is an algorithm
- Debugging
- Composite data types such as lists
- User interaction (input function)
- Visual output (graph plotting).

(Bosch, 2021a, p. 20)

Another result of the project was an overview of the TLEs used in the lecture plans. All lecture plans use text-based programming in Python on personal computers, one computer per student. The teachers had chosen different programming environments, including Anaconda Spyder, Jupyter Notebook, PyCharm, Google Colab, and Trinket. In addition to programming, several teachers emphasized using non-digital tools in the lessons. A majority of the lecture plans included handwriting to support the student's programming process. A subset of the lecture plans also included printed materials such as mathematics textbooks, assignments, and worksheets (Bosch, 2021a).

5.4.3 Information about the Investigated Classes

The information included here was provided prior to the classroom realizations by the teachers responsible for the classes investigated in this study. As the teachers responsible for the two classes collaborated in planning their lessons, the following information is common for the two classes.

TLEs
The teachers had chosen to use text-based programming in Python using the Anaconda Spyder programming environment in previous programming exercises with the class.

Prerequisites
Most of the students had no prior programming experience at the beginning of the fall semester of the subject Mathematics 1T. This study was conducted in the spring semester. The preceding fall, the classes had focused on introductory programming for a few lessons. The students had worked on and handed in an assignment where they implemented solving quadratic equations with programming. In the programs, the students applied the concepts of variables, user input, output, and conditionals. They worked with the Python data types string, float, and integer. Some of the students also parsed from the string data type to integers.

Two weeks prior to the realization in the classroom, the classes focused on programming for one more session with a duration of $2 \cdot 45$ minutes. An online programming resource to support Mathematics 1T provided by the Aschehoug publishing house was used in this preparatory session (Aschehoug, n.d.). The students worked independently with a selection of the modules provided in the online resource. The relevant modules concerned repetition of arithmetic operations in Python, user input, program output, variables, and conditionals. New material in the relevant modules included the concepts of definite and indefinite loops. The last module introduced the Python turtle library. Finishing the modules were given as homework for the students.

6 Result of Design Development

In this chapter, the self-designed sequence of lectures is presented. The sequence of lectures is designed to be a didactical situation for the intended target knowledge. Each teaching session contains a devolution phase, an adidactical work phase, and an institutionalization phase, in line with TDS. The design choices, the didactical variables, are mainly founded on the preliminary analyses in Chapter 4 and Chapter 5, and their rationales will be demonstrated throughout this chapter. Relating to DE, this chapter reports on the *a priori* analysis phase, including the hypotheses created about generic and epistemic students' progression toward the target knowledge.

First, the duration of each session in the didactical situation is presented in Chapter 6.1. A detailed description of the intended target knowledge in both programming and trigonometry is provided in Chapter 6.3. Then the main problem for the teaching sequence and an associated solution proposal will be presented in Chapter 6.4. The material milieu is common for all three teaching sessions and is presented in Chapter 6.5. The main problem is divided so that the students progress towards the target knowledge through the three sessions. An a priori analysis of the realization of each of the sessions is presented in Chapter 6.6, Chapter 6.7 and Chapter 6.8, respectively.

6.1 Duration

The lecture design includes three teaching sessions, adding up to $5 \cdot 45$ minutes. All sessions are intended to be carried out in sequence and completed within one week of classes in the subject Mathematics 1T. Table 1 displays the distribution of minutes in each session. The total duration and division of sessions were customized to fit the class in which the lectures were realized.

Teaching session	**Duration (min)**
1	$1 \cdot 45$
2	$2 \cdot 45$
3	$2 \cdot 45$
Sum	$5 \cdot 45$

Table 1: *Duration of the teaching sessions in the didactical situation.*

6.2 Connection to the Curriculum

The didactical situation addresses two competence aims from the Mathematics 1T subject curriculum:

- The student is expected to formulate and solve problems through the use of algorithmic thinking, different problem-solving strategies, digital tools, and programming
- The student is expected to explain the definitions of sine, cosine, and tangent and use trigonometry to calculate the length, angles, and area of random triangles

(Norwegian Directorate for Education and Training, 2020b, p. 5)

6.3 Target Knowledge

The intended target knowledge k for the sequence of didactical situations designed is dual between programming k_1 and mathematics k_2.

6.3.1 Programming Component of the Target Knowledge

Regarding programming, the target knowledge k_1 is for students to apply subroutines in problem-solving with programming. The aim is for students to implement subroutines in Python appropriately to solve different instances of a family of related problems. Implementing both functions and procedures is included in the target knowledge.

6.3.2 Mathematical Component of the Target Knowledge

In mathematics, the target knowledge k_2 is for students to apply trigonometry to formulate and solve problems in a new situation. The students should understand which of the mathematical sentences in S are applicable for solving the triangles included in a problem, where S consists of The Law of Sines, The Law of Cosines, and The SAS Theorem Area Formula. Furthermore, the students should evaluate which of the sentences in S are most appropriate to use in the problem. Lastly, the students should carry out the application of selected sentences in S to solve the problem.

6.4 Main Problem and Solution Proposal

In line with TDS, the didactical situation was centered around a main problem to be solved by the students. The didactical situation was designed to be a fundamental situation for the target knowledge, meaning that the target knowledge needs to be applied for the students to solve the problem optimally.

6.4.1 Main Problem

The main problem in this didactical situation was for students to create an art program that drew triangles. The graphics should be created using the Python turtle library. The triangles should be non-congruent and drawn based on various combinations of SAS information. The triangles should be drawn at various locations on the turtle canvas until the total area of the triangles exceeded the maximum area decided by a user.

6.4.2 Art Program Solution Proposal

The artifact embodying the solution to the problem was a runnable Python program that produced an artwork. The program should fulfill requirements connected to the target knowledge to be an *optimal* solution. A solution proposal was created to envision the endpoint of the students' progression in the problem-solving process. This solution proposal will further be referenced as the Art Program Solution Proposal, the APSP.

A flowchart representation of the APSP is included in figure 6. The flowchart representation is independent of the programming language implementation. The purpose of the flowchart representation is to highlight the main components of the APSP and the dependencies between them.

One pattern to be recognized in the APSP is the need to repeatedly evaluate the accumulated total area of triangles. The evaluation of the total area is connected to a sequence of steps to be executed if the evaluation stop criterium is not reached. An appropriate decomposition of the evaluation cycle is also visualized through the flowchart representation. Self-contained sub-problems are delegated to their respective subroutines or one-line instructions. The order in which the subproblems are solved is of importance. On the flowchart representation level, the identified sub-problems in the evaluation cycle are:

1. Moving the turtle to a new position
2. Drawing a triangle based on SAS

(a) Calculating the third side length

(b) Calculating the second angle

3. Calculating the area of the associated triangle

4. Updating the total area of triangles

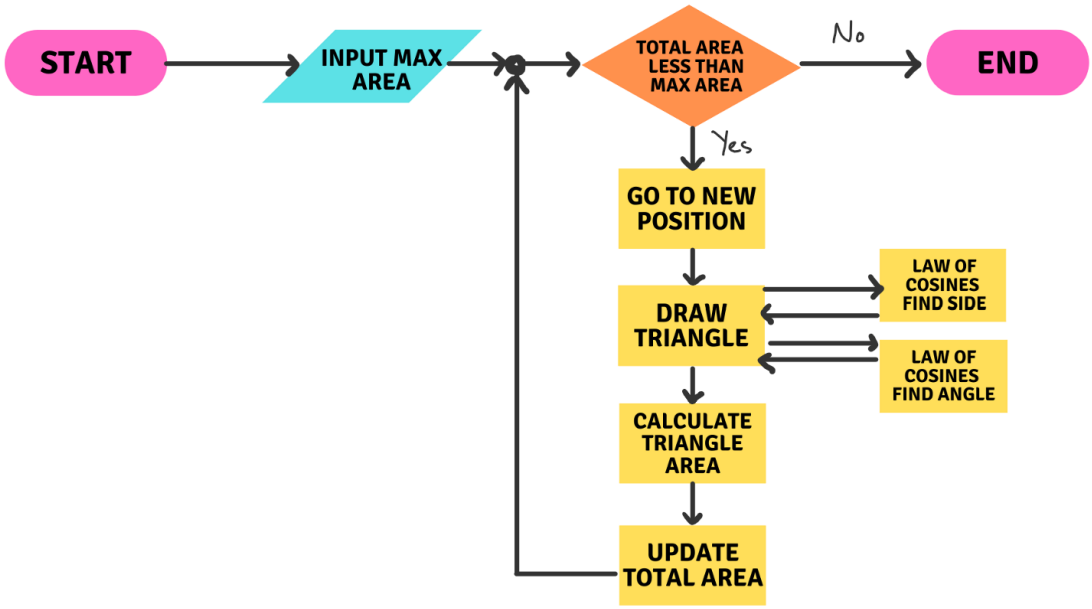


Figure 6: *A flowchart representation of the Art Program Solution Proposal (APSP). Note.* The ellipses symbolize the start and endpoints of the program. The parallelogram symbolizes input to the program. The rectangles symbolize subroutines and one-line instructions solving sub-problems in the program. The rhombus symbolizes a control flow decision based on a variable state evaluation.

The APSP contains an indefinite loop control structure for total area evaluation and execution of the associated subroutines. The rationale behind the indefinite loop control structure is to evoke the need for target knowledge in trigonometry and programming. The intent was that the problem and the APSP that the students work towards have synergy between the mathematical and programming aspects, enabling the didactical situation to be a fundamental situation for the target knowledge.

As specified in the main problem, the subroutine drawing a triangle takes in side-angle-side (SAS) as arguments. Drawing non-right triangles was planned to lead students to use one or more of The SAS Theorem Area Formula, The Law of Sines, and The Law of Cosines to solve the triangle rather than the Pythagorean theorem. Furthermore, it was planned that generalizing the program to draw different triangles in every loop iteration would make the students need to evaluate all possible combinations of triangles. In other words, the target knowledge within trigonometry and the preliminary analysis of trigonometry were part of the rationale behind choosing an indefinite loop as part of the APSP. An indefinite loop was planned to facilitate drawing different triangles in each iteration.

Drawing different triangles in each indefinite loop iteration was also related to the target knowledge in programming and the preliminary analysis of programming. The origins of subroutines set the stage for creating a fundamental situation where implementing subroutines in student programs is the optimal solution. Firstly, the APSP was designed to contain repetition through drawing *multiple* triangles. The frequent need for an instruction sequence was one of the original motivations for the concept of subroutines (Dasgupta, 2014), as described in Chapter 5.2.1. Consequently, the design choice of including repetition in the problem given to the students was to evoke the need to delegate triangle drawing to a subroutine.

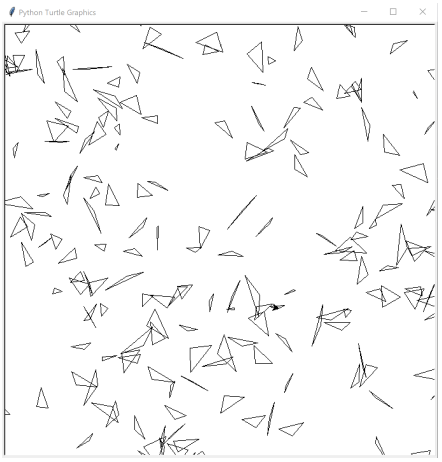
Furthermore, a design choice in the problem given to the student was drawing a set of multiple triangles that were *non-congruent*. As described in Chapter 5.2.1, it is inherent in the definition of subroutines that they are general enough to solve a family of related problem instances (Dasgupta, 2014). A set of non-congruent triangles implies that all three angles and side lengths are not the same in all triangles (Weisstein, n.d.-b). Therefore, drawing the set of non-congruent triangles constitutes a family of related problems where angle sizes and side lengths vary. Hence, drawing non-congruent triangles was a design choice made to evoke the need for including one or more parameters in the triangle drawing subroutine.

Alternative Repetion Constructs
It is worth discussing the alternatives to including the indefinite loop evaluation cycle. A definite loop could be applied to repeat the execution of a set of subroutines. The number of loop iterations in a definite loop is predefined, in contrast to the indefinite loop. Given that the students would be unfamiliar with recursion, the most likely alternative to using an indefinite loop would be a definite loop. When using a definite loop, the number of iterations, and thus the number of drawn triangles, must be specified in advance. Consequently, the area of each triangle must be calculated in advance. The combination of the information SAS must thus be predefined for each triangle, stored temporarily, and then sent into the definite loop so that the predefined triangles can be drawn. It is possible to implement, but temporary storage requires a data type storing the set of information SAS for each triangle and storing the set of all triangles. The students would not be familiar with composite data types at this point. Therefore, it is reasonable to assume that a definite loop implementation by a student would imply drawing the same triangle repeatedly rather than different triangles. Consequently, it was planned to guide the students toward implementing an indefinite loop.

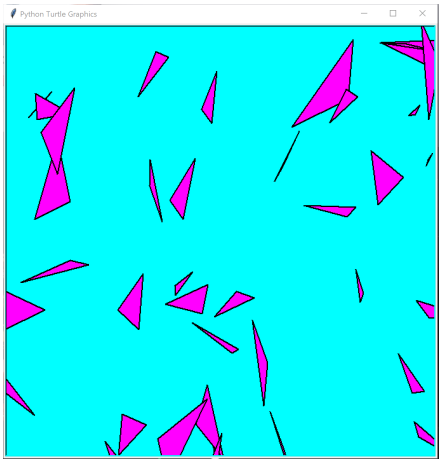
Python Implementation
As the students would write their programs in Python, a Python implementation of the APSP was created. See Figure 7. The subproblems in the APSP can be identified in the Python implementation. The repeated total area evaluation is implemented in lines 54 through 64. Lines 56, 57, and 58 instantiate variables holding the SAS information for a triangle. Five subroutines were implemented. They solve the subproblems of 1) calculating the third side length in a triangle based on SAS, 2) calculating the second angle in a triangle based on SSS, 3) drawing a triangle based on SAS, 4) calculating the area of a triangle using the SAS Theorem Area Formula, and 5) Moving the turtle to a random position on the canvas. The subroutines were included in the program prior to their calls, in line with Python. Running the Python implementation of the APSP produces an artwork in turtle graphics. Figure 8 illustrates two possible artworks created from running the Python implementation of the APSP.

```python
import turtle
import math
import random


def law_of_cosines_side(leg1, angle, leg2):
    angle_rad = math.radians(angle)
    cos_v = math.cos(angle_rad)
    squared_opposite = leg1 ** 2 + leg2 ** 2 - 2 * leg1 * leg2 * cos_v
    opposite = math.sqrt(squared_opposite)
    return opposite


def law_of_cosines_angle(opposite, leg1, leg2):
    cos_v = (opposite**2 - leg1**2 - leg2**2)/(-2*leg1*leg2)
    angle_rad = math.acos(cos_v)
    angle = math.degrees(angle_rad)
    return angle


def draw_triangle(AB, A, AC):
    # Calculates angle C and side BC using the law of cosines
    BC = law_of_cosines_side(AB, A, AC)
    C = law_of_cosines_angle(AB, AC, BC)

    # Drawing the triangle, starting in point B
    turtle.pendown()
    turtle.forward(AB)
    turtle.left(180 - A)
    turtle.forward(AC)
    turtle.left(180 - C)
    turtle.forward(BC)


def triangle_area(side1, angle, side2):
    # Converts the angle from degrees to radians
    angle_rad = math.radians(angle)
    sin_angle = math.sin(angle_rad)
    area = 0.5 * side1 * side2 * sin_angle
    return area


def goto_random_position():
    # Generates random x and y coordinates within the canvas
    x = random.randint(-400, 400)
    y = random.randint(-400, 400)

    turtle.penup()
    turtle.goto(x, y)


max_area = int(input("Input the maximum area: "))

total_area = 0
while total_area < max_area:
    side_length1 = random.randint(10, 40)
    side_length2 = random.randint(10, 40)
    angle = random.randint(1, 179)

    goto_random_position()
    draw_triangle(side_length1, angle, side_length2)
    one_triangle_area = triangle_area(side_length1, angle, side_length2)

    total_area += one_triangle_area
```

Figure 7: *Python implementation of the Art Program Solution Proposal (APSP).*



(a) *Artwork 1. Note.* Triangle side lengths set to between 10px and 40px.



(b) *Artwork 2. Note.* Colors added using subroutines from the Python turtle library. Triangle side lengths set to between 10px and 100px.

Figure 8: *Two artworks produced from running the Python implementation of the Art Program Solution Proposal (APSP).*

6.5 Milieu

The students interacted with the milieu components in the adidactical working periods. The milieu consisted of two pair programming partners, a debugging poster, an assignment sheet, a Python cheat sheet, paper and pencil, and one computer per student for each pair of students working together. A model of the milieu is included in Figure 9.

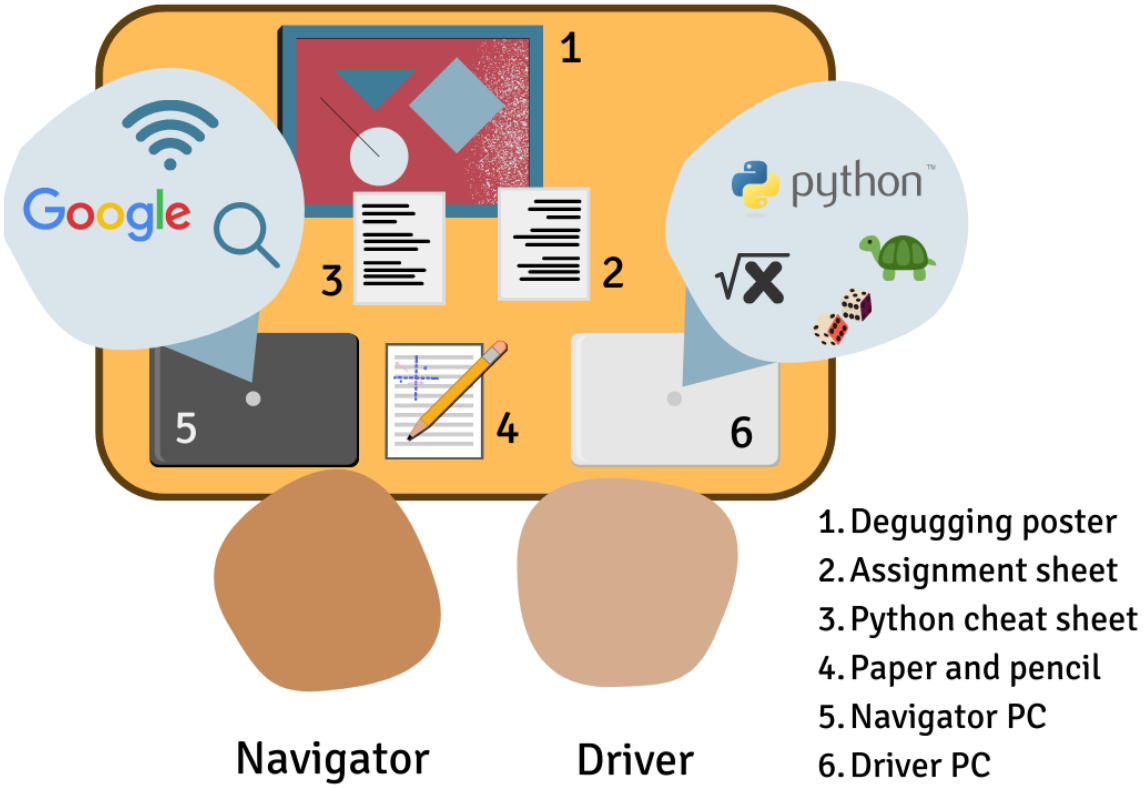


Figure 9: *Model of the milieu in the didactical situation.*

6.5.1 Debugging Poster

Debugging posters were part of the material milieu. Each student pair were provided with a poster on their desk. The poster displays a step-wise process for systematic debugging and aims to support the students in their self-reliant debugging. The choice of including a poster was founded on the results from Michaeli and Romeike (2019b), as described in Chapter 5.3.3. The poster was translated from English to Norwegian and adapted from three cycles to two cycles for simplicity. Specifically, the cycles for compile-time errors and runtime errors were combined into one cycle. The poster was extended to include concrete debugging strategies, as suggested by Michaeli and Romeike (2019b). The concrete debugging strategies included are listed below and were inspired from Li et al. (2019) and Rother (2017).

- Read the line included in the error message and the line above. Look for typos.
- Copy and paste the error message in Google and search. Stackoverflow often provides good answers.
- Read the documentation for Python of the library you are using. How does the built-in function work?
- Test the behavior of parts of the code in a separate file little by little. Work systematically.
- Comment on parts of the code and run the rest of the code. Work systematically.
- Use print() to print out the state of variables as the program executes.
- Trace the relevant code lines carefully in the order they execute. Explain out loud to your partner what happens in each line.
- Send in input where you know the expected output. What is the error now? Do this test repeatedly with different inputs and look for patterns in the errors.

Each unique step in the cycle was given a symbol next to the textual description. The symbols aimed to link the concrete debugging strategies to specific steps in the systematic debugging process. The list of concrete debugging strategies was grouped by cycle steps, and the symbol was repeated next to each group. Some strategies were listed repeatedly, as they were considered relevant for several cycle steps. For example, printing variable state in the program was listed as a strategy for locating the error and determining the error. The introduction of symbols to connect concrete debugging strategies to their appropriate cycle step resulted from the first pilot realization. The pilot participants expressed that the poster did not provide proper guidance on suitable concrete debugging strategies for specific cycle steps. Symbols were a design revision to provide the requested guidance. The revised poster, including symbols, was used in the second pilot and the classroom realization and is provided in Appendix H.

6.5.2 Pair Programming

Pair programming was included in the milieu as it has several relevant benefits for the programming component of the target knowledge in this study. As described in Chapter 5.3.2, pair programming can increase the students' persistence in problem-solving and be beneficial for their engagement. Pair programming can also increase the programming knowledge obtained during a teaching session compared to solo programming. To obtain the desired benefits, the students were paired with friends, in line with the recommendations in Campe et al. (2019). The pairing of students was done in advance by the responsible mathematics teachers who know the students. The criteria given to the teachers for pairings was that students have worked well together before and enjoy working together.

6.5.3 Assignments

The main problem was broken down into three teaching sessions with associated assignments progressing towards the APSP. A printed assignment sheet was provided to each of the student pairs in the adidactical working phases of the three sessions. The assignments given in each session are described in detail in the respective session descriptions.

The progression in the designed assignments was influenced by the PRIMM method, described in Chapter 5.3.1. The assignments progress through the three sessions, from program tracing to making their own art programs. The predict, run, and modify stages were included in the first session. In the second session, the modify, investigate and make stages were included. In the third session, the modify and make stages were included. The PRIMM structure was a design choice made to help students gradually build up the required understanding of syntax and programming structures to create complex art programs like the APSP described in Chapter 6.4.2.

6.5.4 Python Cheat Sheet

A Python cheat sheet was designed as a part of the milieu. Each student pair were provided with a sheet during the working periods. The Python cheat sheet contained a selection of relevant functions for the student programs and associated function documentation. There were four sections on the sheet. The first section included built-in functions in Python, such as round() and int(). The three following sections included functions from the turtle, math, and random libraries, respectively. Examples of library functions included were turtle.speed(), math.sqrt() and random.randrange(). The Python Cheat Sheet is included in full in Appendix I.

The design choice of providing a Python Cheat Sheet was related to the PRIMM method for structuring assignments, as elaborated in Chapter 5.3.1. In the investigation stage, the students work on their program comprehension. The Python Cheat Sheet was a resource to support the students' understanding of their own programs and examples. The Python Cheat Sheet was also a resource to support the modification and make stages by reminding the students of known functions and displaying new possibilities.

In the math library section of the cheat sheet, a conversion from degrees to radians was included to scaffold the students. This design choice was founded on the preliminary analysis of trigonometry.

6.5.5 Paper and Pencil

The students were used to solving triangles by hand calculations carried out on paper. The student with the navigator role in the pair had access to paper and a pencil for performing hand calculations and drawing in the adidactical working phase.

6.5.6 Driver PC

The student with the driver role in the pair had access to a computer during the entire adidactical working phase. Anaconda Spyder was the Python programming environment used by the driver to write code. The Anaconda Spyder programming environment was chosen because the students had used it before and were familiar with it.

The Python math library (Python Software Foundation [PSF], n.d.-a) was included to enable students to use mathematical functions such as square root, sine, cosine, inverse sine, and inverse cosine in their programs. Square roots are needed in the Pythagorean theorem, and the trigonometric functions with their inverses are relevant in the process of solving triangles, as described in Chapter 4.

The Python turtle graphics library (PSF, n.d.-c) was included as part of the milieu available on the driver's computer. The turtle library was included for bridging programming and trigonometry by

visualizing the solving of triangles through drawing them.

The Python random library (PSF, n.d.-b) was included to enable students to draw non-congruent triangles. Side lengths and angles can be generated randomly with functions such as randrange() from the random library.

6.5.7 Navigator Resources

The student with the navigator role in the pair had access to the Internet for looking up relevant resources and documentation in the adidactical working phase. For example, the navigator could use the computer to search Google or the Stack Overflow forum[3] for help.

6.5.8 Intellectual Milieu

The intellectual milieu is the set of the student prerequisites mobilized in the adidactical work phases of the didactical situation. The expectations are built on the preliminary analyses of trigonometry and programming, and especially the institutional analyses.

Regarding trigonometry, the students are expected to know and previously have applied The Law of Cosines, The Law of Sines, and The SAS Theorem Area Formula in traditional mathematics assignments such as solving triangles. The students are expected to know and have applied the triangle and unit circle definitions of $\sin(x)$ and $\cos(x)$ in hand calculations. The students are expected to be familiar with drawing figures to aid calculations in trigonometry. The students are expected to use degrees as the angle measuring unit in calculations with angles. They are expected not to be familiar with radians.

Regarding programming, the students are expected to know and previously have applied the programming concepts of variables, conditionals, definite loops, and indefinite loops. The students are expected to have implemented these programming constructs in Python and consequently be familiar with the syntax. The students are expected to be familiar with primitive data types such as strings and integers. The students are expected to be familiar with common Python built-in functions such as int(), print(), input(), and round(). The students are expected never to have encountered subroutines in programming. The students are expected never to have heard of systematic debugging or pair programming.

6.6 Session 1

The first session's purpose would be to set the stage for the following sessions by equipping the students with the necessary prerequisites. In line with TDS, the students would need to get familiar with the milieu before working self-reliantly on a problem. Getting familiar with the milieu includes knowledge of the resources available for interaction and the associated rules for interaction (Brousseau, 2002). This is part of the devolution phase in the didactical situation where a problem is handed over to the students (Brousseau, 2002, p. 230). The pair programming collaboration and the debugging poster would be new for the students, which meant the possible actions and the rules would not be self-explanatory. Therefore, the first session would be devoted to building up the student prerequisites for the two following sessions.

I planned the following learning outcomes from the first session:

- The students should know what pair programming is. They should understand the responsibilities involved in the driver and navigator roles and have tested both roles.
- The students should know what debugging is. They should know that the poster is an available resource to support their debugging, and they should have tested the steps on the poster while programming in this session.

[3] https://stackoverflow.com/

- The students should know what the Python turtle library is. They should be familiar with central functions in the turtle library and have tested importing the library and used several functions for drawing geometric shapes on the canvas. The students should know that the Python Cheat Sheet is a resource handed out to support the use of the turtle library.
- The students should know the necessary boilerplate functions turtle.done() and turtle.bye() to use the turtle library with the Spyder programming environment. The students should have implemented the boilerplate code in their own programs.
- The students should know that programming is a process that includes understanding the problem, devising a plan, implementing the plan, debugging the program, and evaluating the result. The students should have gone through all the steps in the programming process.

6.6.1 Devolution

Both the overall problem for the week and the session assignment would be handed over to the students in the first session. I planned to explain that the students would create art with programming and trigonometry throughout the week. An example artwork by Johnson (n.d.) would accompany the introduction to illustrate the professional use of programming and trigonometry.

I would explain to the students that programming is a process that consists of planning, implementing, testing, and debugging. I planned to emphasize how frustration and errors are natural parts of the programming process and that the students should expect to meet some difficulties. I would introduce the debugging poster as an aid to help the students in their error handling. I would show the debugging poster and explain that debugging is fixing errors in the code. I planned to explain the different types of errors and relate the two cycles on the poster to the error types. I would encourage students to test out the cycles if they got stuck during the session.

Then, I would organize the class into the planned pairs and introduce the pair programming methodology, including the driver and navigator roles. I would emphasize that both partners are equally in charge of and responsible for the program being developed. Two rules for interaction in the pair would be presented to the students. Firstly, the navigator is not allowed to grab the driver's computer keyboard or mouse. Instead, the navigator has to explain their ideas in words to the driver. Secondly, the driver is expected to think aloud and explain to the navigator what they are doing. In the case of an odd number of students, I would divide three students into a triplet with one driver and two navigators.

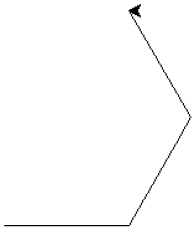
Following the introduction of the milieu, the devolution would transition into the adidactical work phase. I would display a Python program using turtle functions to the students. See Figure 10a. I would ask the students to predict the output of the program together with their partners before proceeding to the following assignments. I would hand out the printed materials. Now the problem would be handed over to the students.

6.6.2 Adidactical Work Phase

The program was designed to encourage the students to understand and modify it. I planned for the students to believe that the program created an equilateral triangle due to the 60° angle in line 5. As can be seen from Figure 10b, the program output is not an equilateral triangle. The following assignment for the students would be to modify the program to fit their original hypothesis. If their hypothesis was right in the first place, they would jump straight to the next assignment.

```
1  import turtle
2
3  for n in range(3):
4    turtle.forward(100)
5    turtle.left(60)
6
7
8  turtle.done()
9  turtle.bye()
10
```

(a) *Program.*



(b) *Output.*

Figure 10: *The Python program displayed to the students in the devolution phase in session 1 and the resulting output in the turtle graphics window.*

The following assignment would include further modification of the program. The assignment would ask the students to plan, implement and test their modifications. An example idea of creating a program that draws 100 blue octagons would be included to spark the students' ideas. The included example's purpose was to encourage the implementation of colors from the turtle library and loops in the student programs. Appendix J includes the assignments given to the students in Norwegian for all three sessions.

Teacher Regulation
During the adidactical work phase, I would expect the students to encounter errors and ask me for help. I planned to acknowledge their frustration and listen to their questions but not give them the answers. I planned to demand autonomy from the students to foster self-reliant debugging, in line with the findings from (Michaeli & Romeike, 2019a). I would refer to the navigator's responsibility for helping the driver. I would also remind the students of the aids permitted, such as the debugging poster and the Python Cheat Sheet.

6.6.3 Institutionalization

I would end the first session by decontextualizing the programming process from the situation. I would emphasize how the students would benefit from planning their programs prior to implementation in the following session. I would also emphasize how the students would benefit from using the debugging poster and the systematic debugging process for handling errors throughout the week. I would highlight honorable student actions such as drawing on paper for support or creating a plan prior to implementation.

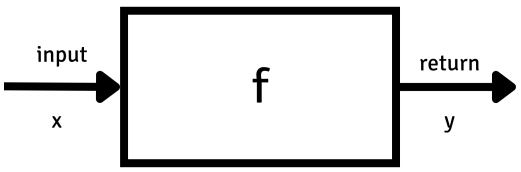
6.7 Session 2

The second session would have a dual purpose between programming and trigonometry. One aim would be for the students to know how to declare subroutines with and without parameters in Python and call the subroutines within their programs. Regarding trigonometry, the aim would be for students to solve triangles based on SAS information. By combining the two, the main objective of this session would be for the students to produce a program that draws a triangle using a self-made subroutine with parameters for the information provided in SAS.

6.7.1 Devolution

I would build from the students' existing knowledge about mathematical functions to introduce subroutines. I would write an example of a mathematical function, $f(x) = x^2 = y$, on the whiteboard. I would draw an abstraction of the function, highlighting the function name f, the parameter value x, and the return value y. See figure 11a. I would explain that the function f

can be implemented with programming. I would further explain that functions in programming is a wider term than mathematical functions and provide examples in the same abstraction form. I would draw an abstraction of the mathematical area formula as a function with two parameters, baseline b and height h, and the return value area. See Figure 11b. The third abstraction I would draw is a procedure for drawing a triangle with no parameter values in Figure 11c. The last abstraction I would draw would be a procedure for drawing a triangle taking the parameter value. See figure 11d.



(a) *Function for calculating the value of $f(x)$. Note.* The function takes one parameter, x, and returns one value, y.

(b) *Function calculating the area of a triangle. Note.* The function takes two parameters, b and h, and returns one value, area.

(c) *Procedure for drawing a triangle. Note.* The procedure takes no parameters.

(d) *Procedure for drawing a triangle. Note.* The procedure takes one parameter, side_length.

Figure 11: *Illustration of the four subroutine abstractions intended drawn to students in the devolution phase in session 2.*

I would follow up the abstractions on the whiteboard with a live coding session where I would implement the abstractions in Python. The demonstration would include how to declare subroutines. I would demonstrate how to include parameter values and how to include a return value. I would show how to call subroutines. The end product of the live coding session is included in Appendix K.

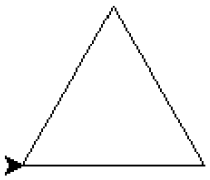
I would end the live coding session with a transition to the adidactical work phase. I would explain that the students were going to implement subroutines that drew triangles, focusing on subroutines without a return value.

6.7.2 Adidactical Work Phase

The first assignment would build on the programs written in session 1. The students would be provided with a program that draws an equilateral triangle. See Figure 12. The first assignment would be to wrap the program in a function. In the following assignment, the students would create a new function that draws a different triangle than an equilateral. The choice of a triangle would be up to the students. By asking the students to draw a different triangle, the aim would be to evoke the need to apply their mathematical knowledge about triangles and apply their newly acquired knowledge about how to implement subroutines in Python. The choice of a triangle would be left to the students to accommodate differentiation.

```
import turtle

for n in range(3):
  turtle.forward(100)
  turtle.left(180 - 60)


turtle.done()
turtle.bye()

```

(a) *Program.*



(b) *Output.*

Figure 12: *The Python program provided to the students in the first assignment in the adidactical work phase in session 2 and the resulting output in the turtle graphics window.*

The students' subroutine for drawing a triangle would be modified to include at least one parameter. I planned the inclusion of parameters to evoke a need for solving a more general problem, namely drawing similar triangles that are non-congruent.

The last assignment would be to let two sides and the angle in between them (SAS) be the parameters. This assignment aimed for students to apply the law of cosines to solve the triangle.

Swapping programs
The student groups would swap programs with another group during the adidactical work phase. The intention was to increase the focus on code comprehension, in line with the interpretation stage in PRIMM (Sentance et al., 2019a). The questions given to the students would include comprehension of different abstraction levels of the peer group's program. One question would be to formulate the differences between their own program and the peer group's program. The questions are included with the assignments in Appendix J.

Teacher Regulation
I would demand self-reliance in debugging in this adidactical work phase, just like in the first session. Due to the mathematical focus of the assignments, I would guide the students to sketch and perform hand calculations on paper. For students struggling with implementing a function to draw a different triangle, I would advise them to start with a right triangle, as the students would be familiar with solving triangles with the Pythagorean theorem.

6.7.3 Institutionalization

I would end the adidactical working phase by institutionalizing the use of subroutines. I would emphasize that functions are programming constructs used to store a sequence of instructions for reuse. Furthermore, I would highlight how subroutines taking one parameter provide the opportunity for drawing similar triangles. I would demonstrate drawing similar triangles with a live coding example. I would implement a Python function drawing right triangles taking one side length as a parameter. I would include the inverse sine function from the math library for solving the triangle. The end product of the live coding session is included in Appendix K.

6.8 Session 3

The aim of the third session would be for the students to end up with programs similar to the APSP Python implementation. By progressing through the assignments culminating in creating a program similar to the APSP, the aim would be for the students to have obtained the target knowledge in trigonometry and programming.

6.8.1 Devolution

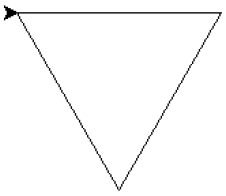
The devolution of the third session would be rather short to provide the students with enough time in the adidactical work phase. I would tell the students that in this last session, everything they have done in the two previous sessions will become useful. I would tell them that today they will create artworks using the subroutines they created in session 2 that draws triangles. I would tell them that if their delivery from session 2 did not work, an alternative subroutine would be provided on the assignment sheet for them to use. I would remind the students of the pair programming methodology and organize the class in their pairs. I would remind and advise the students to plan their programs in advance and that programming is a process that consists of planning, implementing, testing, and debugging. I would remind the students of all the materials available in the milieu. Lastly, I would hand out the assignments, the debugging poster, and the Python Cheat Sheet to the students.

6.8.2 Adidactical Work Phase

The students would start out with their subroutines for drawing one triangle. Figure 13 displays the code provided to the students on the assignment sheet and the output of the code. The code was provided to enable all student pairs with the same starting point in session 3, also the students that did not produce a properly functioning subroutine for drawing a triangle in session 2.

```
1  import turtle
2
3  def likesidet_trekant(sidelengde):
4    for n in range(3):
5      turtle.forward(sidelengde)
6      turtle.right(180 - 60)
7
8
9  likesidet_trekant(150)
10
11 turtle.done()
12 turtle.bye()
13
```

(a) *Program.*



(b) *Output.*

Figure 13: *The Python program provided to the students in the first assignment in the adidactical work phase in session 3 and the resulting output in the turtle graphics window.*

The student pairs would modify their programs to draw several triangles at different locations on the canvas. Then, they would modify their programs to draw exactly 10 triangles at different locations on the canvas. These assignments were designed to evoke the need to simplify repeated function calls with a loop. The students would further modify the program to draw triangles until the total area of triangles exceeds 10 000 square pixels. The formulation of this assignment had a dual purpose. The first purpose was to evoke the need to calculate the area of the drawn triangles. The intention would be for the students to figure out that determining and drawing their triangle and calculating the area of said triangle would be easiest to connect by using SAS. Secondly, the word "until" was included to evoke the implementation of repeated evaluation of the total area with a while-loop. The following assignment would be to modify the program so that the user could decide the maximum total area of the triangles. This assignment was planned to eradicate all attempts to implement the assignments with a pre-determined number of triangles, and evoke the need for a while-loop. Then, the students would bring the work of art to life with colors. The last assignment would be to substitute the subroutine drawing one triangle. This assignment was planned to ensure the decomposition of area calculation and drawing triangles in separate subroutines, and to evoke the need to calculate the area based on the SAS Theorem Area Formula.

Teacher Regulation
I would expect the student to encounter numerous errors in this session, as several programming

constructs such as subroutines and loops would be implemented. I would demand self-reliance in debugging, as in sessions 1 and 2. To help the students stuck on errors, I would advise them to use the systematic debugging process and the related strategies provided on the debugging poster. I would help the students to identify which cycle and step in the cycle they are stuck on, and suggest an appropriate strategy to test. I would also remind the students to devise a plan, to draw sketches, and to apply the trigonometric theorems they are familiar with if they would express not knowing how to start.

I would expect the students to evaluate all three trigonometric theorems available in their implementations of a new subroutine for drawing a triangle. As described in the preliminary analysis of trigonometry, The Law of Cosines can be applied and reformulated to uniquely determine an angle rather than a side in a triangle, as the $\cos(x)$ function is injective on the domain of angles for a triangle. Due to the $\sin(x)$ function not being injective on the domain of angles in triangles, The Law of Sines can not be applied to uniquely determine an angle. Consequently, I would regulate the students if they would attempt to use The Law of Sines to calculate angles in the triangle to be drawn. I would help the students realize that The Law of Cosines is more appropriate in the situation. I would do so by asking the students to draw an obtuse triangle with their program. As the math library implementation of the asin(x) function only outputs acute angles, their program would not draw a triangle.

6.8.3 Institutionalization

I would institutionalize the whole didactical situation by summarizing what we have done during the three sessions. I would explain how pair programming and debugging are methodologies applicable in all programming settings.

I would display the interactive artwork by Johnson (n.d.) again. The purpose of displaying the professional artwork again would be to help the students decontextualize the use of subroutines and trigonometric theorems from the didactical situation, and see the value outside of drawing triangles. I would emphasize how the professional artwork applies trigonometry to create circular motion rather than triangles. I would close the tab with the professional artwork to create further distance from art production. Then, I would emphasize how the application of subroutines is useful when programming in contexts other than art production. I would do so by drawing the same abstractions of subroutines as in session 2 on the whiteboard. I would describe applications of the different types of subroutines outside of the art production domain.

7 Results from the Classroom Realizations

The results from the classroom realization are presented in this chapter with supporting excerpts from the data material. The data analyzed was observational data from one classroom, student programs from sessions 2 and 3, the in-vivo analysis between sessions, and the post-realization student interviews.

7.1 Session 2

The majority of student groups chose to create programs that drew a right triangle in session 2. The right triangle programs typically included an application of the Pythagorean theorem for calculating the hypotenuse or one of the catheti. None of the right triangle student programs included the definition of sin or cos for solving the triangle.

7.1.1 Hard-Coding Triangles

One implementation variation was drawing a right triangle using a function taking one side length as a parameter. However, the remaining two side lengths were hard-coded into the function. Consequently, a triangle would only be drawn for one specific parameter value. An example from student pair 2 in class 1 is included in Figure 14, in which case the program draws a triangle only when the value 100 is passed as an argument.

```
1  import turtle
2
3  def trekant(sidelengde):
4      turtle.left(90)
5      turtle.forward(sidelengde)
6      turtle.left(-120)
7      turtle.forward(200)
8      turtle.right(150)
9      turtle.forward(175)
10
11 trekant(100)
12
13 turtle.done()
14 turtle.bye()
15
```

Figure 14: *Hard-coding triangles. Note.* The program draws a triangle only when the value 100 is passed as an argument to the subroutine "trekant" (triangle). Delivered after session 2 by student pair 2 in class 1.

7.1.2 Special Cases of Triangles

Other variations implemented drawing special cases of right triangles, such as a 30-60-90 triangle. Unique to 30-60-90 triangles is the ratio between the shortest cathetus c and the hypotenuse. The hypotenuse length is $2 \cdot c$ (Weisstein, n.d.-a). The shortest cathetus was passed as a parameter to the function. Another special case triangle in the student programs was drawing an isosceles right triangle, where both catheti have the same side length. The side length was passed as an argument in a function. Examples from student pair 7 in class 2 and student pair 8 in class 1 are included in Figure 15. Worth emphasizing about these programs is that they correctly drew similar triangles, opposed to the program in Figure 14. The functions created by the students displayed in Figure 15 took one side length as parameter and based calculations for the remaining side lengths on the parameter.

7.1.3 Redundant For-Loops

Figure 15 also illustrates another strong tendency in the student programs. A majority of student programs included redundant for-loops for completing one iteration of an instruction sequence.

```
1  import turtle
2  import math
3  turtle.speed(0)
4
5  def rettvinklettrekant(sidelengde):
6     turtle.forward(sidelengde)
7     turtle.left(120)
8     H=2*sidelengde
9     turtle.forward(H)
10    turtle.left(150)
11    a=H**2-sidelengde**2
12    K=math.sqrt(a)
13    turtle.forward(K)
14
15 rettvinklettrekant(50)
16
17
18 turtle.done()
19 turtle.bye()
20
```

(a) *Student program drawing a 30-60-90 triangle. Note.* Delivered after session 2 by student pair 7 in class 2.

```
1  import turtle
2
3  def rettvinklet(sidelengde):
4      for n in range(1):
5              turtle.forward(sidelengde)
6              turtle.left(90)
7              turtle.forward(sidelengde)
8              turtle.left(180-45)
9              import math
10             a=sidelengde**2+sidelengde**2
11             turtle.forward(math.sqrt(a))
12 rettvinklet(150)
13
14 turtle.done()
15 turtle.bye()
16
```

(b) *Student program drawing an isosceles right triangle. Note.* The program also includes a redundant for-loop. Delivered after session 2 by student pair 8 in class 1.

Figure 15: *Student programs drawing special cases of triangles.*

A handful of student pairs started the final assignment in session 2. The last assignment was solving and drawing a triangle using the information side-angle-side (SAS). One of the pairs applied The Law of Cosines for calculating the opposite side of the known angle, as can be seen in line 22 in Figure 16a. An observation about this program was that it included a logical error. In line 22, the value for an angle in variable theta was calculated using the built-in functions sin() and asin() from the math library. The angle v was passed as an argument. The trigonometric functions in the math library take the angle parameter in radians (Python Software Foundation [PSF] ,n.d.-a), whereas v was passed in degrees. Consequently, the wrong angle was rotated in line 23, and a triangle was not drawn.

Another pair attempted to implement the last assignment using the SAS Theorem Area formula. The program is included in Figure 16b. As shown in lines 7 and 8, the students converted the parameter angle A from degrees to radians before calculating the value sin_A with the math library.

```python
#Importerer turtle og math bibliotekene
import turtle
import math

turtle.speed(1)
turtle.hideturtle()

#Sidelengder
side1 = int(input("Sidelengde 1: "))
side2 = int(input("Sidelengde 2: "))
hypotenus = math.sqrt(side1**2+side2**2)
vinkel = int(input("Skriv inn en vinkel: "))

#Tegner en rettvinklet trekant
def Trekant(s1, s2, v):
    turtle.forward(s1)
    turtle.left(v)
    turtle.forward(s2)

    h = math.sqrt(s1**2+s2**2-2*s1*s2*math.cos(v))

    theta = math.asin(s1*(math.sin(v)/h))
    turtle.left(180 - theta)

    turtle.forward(h)

Trekant(side1, side2, vinkel)

turtle.done()
turtle.bye()

```

(a) *Student program applying The Law of Cosines. Note.* Delivered after session 2 by student pair 5 in class 1.

```python
import turtle
import math

turtle.clear()

def trekant(A, b, c):
    vr=math.radians(A)
    sin_A=math.sin(vr)
    def areal():
            (1/2)*b*c*sin_A


turtle.penup()
turtle.goto(100, 100)


turtle.done()
turtle.bye()

```

(b) *Student program applying the SAS Theorem Area Formula. Note.* Delivered after session 2 by student pair 1 in class 2.

Figure 16: *Student programs drawing triangles bases on SAS.*

7.2 Session 3

7.2.1 Repeated Function Calls

In session 3, the majority of student programs delivered had completed or attempted assignment 4. In this assignment, the students created programs that drew triangles until the total area exceeded 10 000 square pixels. An observation about these programs was the diversity in implementing the repeated drawing of triangles. In one variant, the programs included a function for drawing multiple triangles. A for-loop was included within the function definition, and the function was called once in the program. A second variant wrapped a for-loop around the function declaration and a call to the function. A third variant isolated the drawing of one triangle to a function and called the function repeatedly with a for-loop. A fourth variant omitted a loop and repeatedly called the function with the same arguments. The four variants are included in figure 17.

```
import turtle
import math

turtle.speed(100)

def likesidet_trekant(sidelengde):
  for n in range(360):
    turtle.forward(sidelengde)
    turtle.left(180 - 60)
    turtle.forward(sidelengde)
    turtle.left(180-60)
    turtle.forward(sidelengde)
    turtle.left(180-60+1)

likesidet_trekant(180)

turtle.done()
turtle.bye()

```

(a) *Student program with for-loop included in the function definition. Note.* Delivered after session 3 by student pair 3 in class 2.

```
import turtle

turtle.speed(10)

for n in  range(3):
    def likesidet_trekant(sidelengde):
      for n in range(3):
        turtle.forward(sidelengde)
        turtle.left(180 - 60)

    likesidet_trekant(100)

    turtle.penup()
    turtle.forward(105)
    turtle.pendown()


turtle.done()
turtle.bye()

```

(b) *Student program wrapping the function declaration and call in a for-loop. Note.* Delivered after session 3 by student pair 8 in class 2.

```
import turtle


def likesidet_trekant(sidelengde):
  for n in range(3):
    turtle.forward(sidelengde)
    turtle.right(180 - 60)



for n in range(1,50):

    likesidet_trekant(1*70)
    turtle.forward(90)
    turtle.right(100)
    turtle.speed(1000)
    turtle.left(100)
    likesidet_trekant(1*70)
    turtle.right(100)



turtle.done()
turtle.bye()

```

(c) *Student program isolating the declaration of the function drawing one triangle from the for-loop with repeated function calls. Note.* Delivered after session 3 by student pair 9 in class 1.

```
import turtle

def likesidet_trekant(sidelengde):
  for n in range(3):
    turtle.forward(sidelengde)
    turtle.right(180 - 60)

turtle.speed(0)
turtle.shape("turtle")
turtle.color("magenta")
turtle.bgcolor("blue")

likesidet_trekant(150)
likesidet_trekant(300)
likesidet_trekant(170)

turtle.left(300)

...

likesidet_trekant(150)
likesidet_trekant(300)
likesidet_trekant(170)
turtle.left(300)

likesidet_trekant(150)
likesidet_trekant(300)
likesidet_trekant(170)

turtle.penup()
turtle.goto(50,50)


turtle.done()
turtle.bye()

```

(d) *Student program omitting a loop. Note.* Delivered after session 3 by student pair 6 in class 2. The program has been shortened.

Figure 17: *Four variants of student implementations of repeated function calls.*

7.2.2 Calculation of Area

Another observation was that only the few student programs exceeding assignment 4 attempted to calculate the area of a single triangle. Assignment 5 was to let a user of the program decide the maximum area. Furthermore, only one of those programs applied the SAS Theorem Area Formula to calculate a single triangle's area.

Related to the calculation of area, two of the student programs stood out by attempting to keep track of the total area of all drawn triangles. The first program applied a for-loop and drew similar right triangles taking a randomly generated integer as an argument for one side length. The total area was printed at the end of the program. The program draws 15 similar triangles, and the total area varies due to the randomization of side length. The program is included in Figure 18.

```
1  import turtle
2  import math
3  import random
4  turtle.speed(0)
5
6  def rettvinklet_trekant(sidelengde):
7     turtle.forward(sidelengde)
8     turtle.left(120)
9     H=2*sidelengde
10    turtle.forward(H)
11    turtle.left(150)
12    a=H**2-sidelengde**2
13    K=math.sqrt(a)
14    turtle.forward(K)
15
16 samla_areal=0
17
18 for i in range(15):
19     turtle.pendown()
20     b=random.randrange(1,200)
21     rettvinklet_trekant(b)
22     k=math.sqrt((2*b)**2-b**2)
23     areal=k*b*1/2
24     samla_areal=samla_areal+areal
25
26
27     turtle.penup()
28     turtle.goto(random.randrange(-200,200),random.randrange(-200,200))
29
30 print(samla_areal)
31
32 turtle.done()
33 turtle.bye()
34
```

Figure 18: *The first student program keeping track of the total area of triangles. Note.* Delivered after session 3 by student pair 7 in class 2.

The second program that kept track of the total area applied a while-loop. The program is included in Figure 19. A variable named totalareal (total_area) was instantiated with the value 0 in line 15. In line 16, a variable named maks (max) was declared. The max variable stored an input value from the user. A while-loop was then executed if the value of the total area was less than the value of max. The while-loop called the function to draw a triangle with the same argument, attempting to draw congruent triangles. The program only *attempted* to draw triangles due to a runtime error. The calculation of the area in line 26 relied on a variable called sidelengde (side_length) that was undefined.

```
1  import turtle
2  from random import randint
3  import math
4  turtle.speed(10)
5  def likesidet_trekant(sidelengde):
6      turtle.forward(sidelengde)
7      turtle.left(180 - 90)
8      turtle.forward(sidelengde*0.6)
9      h=math.sqrt(sidelengde**2)+((0.6*sidelengde)**2)
10     alpha=math.asin(sidelengde/h)
11     turtle.left(alpha)
12     turtle.forward(h)
13 vinkelr=math.radians(90)
14 sin_90=math.sin(vinkelr)
15 totareal=0
16 maks=int(input("skriv inn maksareal"))
17 while totareal < maks:
18     turtle.color("magenta")
19     turtle.fillcolor("blue")
20     turtle.begin_fill()
21     turtle.pendown()
22     likesidet_trekant(10)
23     turtle.penup()
24     turtle.goto(randint(-300,300),randint(-300,300))
25     turtle.end_fill()
26     areal = (1/2)*sidelengde*(sidelengde*0.6)*sin_90
27     totareal+=areal
28 turtle.done()
29 turtle.bye()
30
```

Figure 19: *The second student program attempting to keep track of the total area of triangles. Note.* Delivered by student pair 1 in class 2.

7.2.3 Variable and Parameter Scope

The program containing a while-loop in Figure 19 was created by Dina and Vegard, the pair observed in class 2. Their rationale behind the undefined variable side_length was accessible from their work dialogue and the associated screen recording. Transcription codes are included in Appendix L.

> Timestamp 28:22
> V: side_length is not defined yet. But I don't think it is a problem.
> D: Did we not define it here? (Dina highlights line 5 in the program.)
> V: I don't know why it-
> D: side_length. Have I typed it correctly? Yes.
> V: I don't think it is a problem yet.
> [...]
> Timestamp 36:32
> V: It is an error message saying undefined side_length.
> D: That side_length is not defined?
> V: Is it not?
> D: I thought we defined it here.
> V: Because in the equilateral_triangle- down here [the function] says what side_length should be. Is that wrong?
> D: And it is weird that [the error message] comes here, and not in any of the other places side_length is used. Did we type it wrong anywhere?

From the dialogue, it seemed like Dina and Vegard believed that the side_length variable was given a value in line 5 or line 22. Line 5 was the first line of the equilateral_triangle function declaration, including a parameter called side_length. Line 22 was a call to the equilateral_triangle function, passing the number 10 as an argument.

7.3 Perceptions of Motivational Factors

The student interviews were opened with questions about the students' overall opinion of the lessons. Without exception, all four students' immediate answers were positive. The students expressed that the lessons had been fun and contributed to their programming learning. Martin from class 2 also emphasized that he expected to make use of the newly acquired skills and knowledge in the future:

> I thought it was very fun, and I learned a lot. It makes me want to learn more. And I know that programming is a good skill to have, at least how the world is becoming in the future. So it's fun that it's being more integrated in school. (Martin, class 2)

When asked to elaborate on what made the lessons fun, the students highlighted the build-up of assignments from examples to exploration with their own programs. Vegard appreciated the increasing complexity:

> [I liked to] make the program more and more complex. When you keep going further and further. You can use things you have created earlier and improve the program. (Vegard, class 2)

Furthermore, getting visual feedback as their product developed throughout the week was motivating:

> You got such a sense of mastery when you managed to get a product that was cool to look at. Like, see what you could do with the different codes. At first, I thought it was a little confusing. [...] but I got a pretty cool result. (Viktor, class 1)

7.4 Ambivalence to Pair Programming

The students were ambivalent about whether or not they perceived pair programming was helpful in the programming process. Vegard reflected on the potential for receiving feedback, but he felt that working alone would enable him to produce a more advanced result:

> I think the concept [of pair programming] might be more effective. That you get, for instance, constructive feedback on your code from another person. Instead of just seeing it through your own eyes. I understand that switching between the driver and navigator roles is for both to be able to program, but it may be a little less effective as well. [...] I think I could have gotten further with the program if I had worked alone because I type faster. [...] But I still think it was pretty fun to [program] in pairs. I just want to do it better. I want to improve my code as much as possible. Working in pairs and having to give instructions to someone else can slow that down a bit. (Vegard, class 2)

Martin perceived the navigator role differently than Vegard. He elaborated on his own learning process from instructing the driver:

> If you have to explain what you understand to someone else, you get pushed to break it down and simplify it. By explaining it in simpler words, you understand it better yourself. (Martin, class 2)

7.5 Debugging Perceptions

The students were asked in the interviews about what they experienced as difficult in the sessions. All the students responded that error handling was frustrating, highlighting different logical errors as particularly frustrating. Vegard shared how he felt stuck when they encountered a logical error:

> The code did not execute the way it was supposed to. It did not run, and we did not get an error message. Everything fell apart because we got no error messages. How are you supposed to improve it? Or fix it. (Vegard, class 2)

Robin had a similar experience:

> First, we almost made a star. It started here, and then it went out like that, and then it went straight in. And then it went straight out. And then again several times. [...]. I do not quite remember what we had written. But that was not what we wanted, because we wanted triangles. We just had to try and fail a few times. (Robin, class 1)

A follow-up question asked the students about whether or not they had used the debugging poster when encountering logical errors. Viktor and Robin replied that they raised their hands instead of using the poster. Martin and his partner had tried to use it once, but they did not manage to implement a fix by using the debugging poster:

> We tried to go through the poster several times to get to the result we wanted, but we had no clue about which new lines to put into the program.

The students were further asked to describe which methods they applied instead of the debugging poster when they encountered errors. Vegard explained how he mostly drew from his prior programming experience:

> I look through [the program] over and over again to find out what is wrong. You know, if you write enough, if you code enough, you get used to recognizing what looks right. Maybe. It gets easier to find the error. And, of course, I read the error messages. You have to be able to interpret them. Often they give you an answer to what is wrong. Also, there is a lot of trial and error. To try something and see if it works or not. (Vegard, class 2)

7.6 Debugging Observations

The students encountered logical errors while programming. The recordings of Dina and Vegard from the adidactical work phases show that they do not employ a systematic debugging process. They rely on error messages and trace the program looking for causes of the error. They also try to apply the debugging poster but quickly discard it. The students attempt these strategies for a short amount of time before asking the teacher for help. The following observation from session 3 documents their actions when encountering a logical error. Transcription codes are included in Appendix L. As seen from the transcript, Vegard introduces a logical error in the while-loop, confusing the symbols $>$ and $<$. The students identify the error by running the program. The students try to debug the program for 40 seconds before raising a hand to ask for help.

> Timestamp 18:07
> V: A while-loop just does something until a condition is met. So while total_area is less than 10 000... (Vegard types while total_area $>$ 10 000).
> [...]
> Timestamp 19:21
> (Vegard runs the program.)
> V: Okay? It did not run. And we're not getting an error message either.
> D: We can look at this then. (Dina laughs and picks up the debugging poster.)
> Timestamp 19:30
> V: Okay. Does the program run without any error messages? Yes.
> Read and understand the error message. It is not-
> D: Isn't it rather: Does the program run without any error messages?

Revert the changes you made in the code.
V: I don't bother to do that.
D: No.
(Dina and Vegard mumble. Vegard traces the program on the screen and mentions the word "radians". Dina reads on the poster. Both students trace the code.)
Timestamp 20:00
(Dina raises her hand, and the instructor comes over.)

7.7 Evaluation of Target Knowledge Attainment

From the results presented about the students' delivered programs, it can be proposed that the designed didactical situation has the potential for students to attain the target knowledge in trigonometry, but that most students did not. In session 2, one of the pairs implemented The Law of Cosines to solve and draw a triangle based on the information SAS. In session 3, one student pair implemented the SAS Theorem Area Formula to calculate the area of each triangle drawn. These deliveries indicate that the students understood the trigonometric component of the assignments and managed to apply their existing knowledge about these trigonometric theorems in the situation. On the other hand, the results also demonstrate how most student pairs' trigonometry applications were on a lower than the intended target knowledge. For instance, the Pythagorean theorem.

There is a similar tendency regarding the target knowledge in programming. Most student pairs managed to declare functions, including parameters, in their programs. Repeated function calls were also implemented in most student programs delivered in session 3. However, their application of subroutines did not reflect the property of solving a family of related problems. Calling a function with different parameter values was only implemented in a handful of the student programs. The students also emphasized functions as valuable for repeating a sequence of instructions in their reflections regarding functions in the interviews, without mentioning solving a generalized problem. These findings indicate potential in the didactical situation for students to obtain the target knowledge in programming. However, the process toward the target knowledge seems restrained for most students.

8 Discussion

This chapter provides a discussion of the results presented in Chapter 7. The discussion is centered around the results' indications of contributing and constraining factors for obtaining the target knowledge, and consequently, not all results are discussed. This chapter reports on the a posteriori analysis and validation of the didactical situation, the fourth phase of DE.

8.1 Students' Progression in the Assignments

Noteworthy about the attainment of the target knowledge in trigonometry and programming is the seeming interdependence between attainment and the progression in the assignments. The assignments were designed to unite solving non-congruent triangles based on SAS with appropriately implementing a generalized solution to the said problem using subroutines. The most advanced student programs include the elements reflecting obtaining the target knowledge in both trigonometry and programming. Interestingly, the inverse also seems to be the case. The majority of students' programs include lower-level trigonometry and lack the generalization characteristic in their subroutines. From these results, it can be argued that the planned progression in the assignments was both a constraining and a contributing factor to the attainment of the target knowledge in both programming and trigonometry. In other words, it seems like only the students who managed to progress to the last assignments benefited from the planned synergy evoking generalization of subroutines and SAS determined triangles. The students whose progression was terminated earlier seem not to have benefited from the synergy. Consequently, the students' possibilities for obtaining the target knowledge in this didactical situation were conditioned by their progression in the assignments. The subsequent discussion of contributing and constraining factors in this chapter is connected to the assignment progression, as it seems like a decisive factor for the students' possibilities for obtaining both target knowledge.

8.2 Engagement

In the interviews, the students shared that they were motivated by seeing their artworks evolve throughout the week. Instant visual feedback on the students' work was enabled with the Python turtle library. The assignments built on the PRIMM structure also enabled the students to get visually pleasing results quickly, which they could gradually develop further through the modify and make stages. The students referred to the produced artworks as *their own* in the interviews, which also indicates a high level of ownership of the products. From these results, it can be argued that the design choices of including the turtle library in the milieu and structuring the assignments in line with PRIMM were contributing factors in the didactical situation toward the target knowledge in programming and trigonometry. The students were having fun, they were motivated, and had a feeling of ownership of the artworks, which again contributed to the high level of engagement and persistence among the students.

Visual feedback from graphics promoting motivation and persistence is in line with the results from Kaufmann and Stenseth (2021), who conducted an experiment with programming in mathematics on the lower secondary level in Norway. The modify stage in PRIMM as an engaging factor is reported by British programming teachers on the lower secondary level. The compliance with literature for the engaging and motivating effects of visual feedback and PRIMM structured assignments support the suggestion to continue these design choices. However, the component of including art production with programming in mathematics is a novel approach in this project that seems promising but requires further research.

8.3 Focus on Visual Results over Process

On the other hand, the same result-focused student statements can also indicate that the students valued working towards visually pleasing results so high that they focused less on the processes

involved in programming that, in the short term, do not contribute directly toward visual results. In the interviews, one of the students expressed that working in pairs slowed him down and that he would have produced a more advanced result alone. Similarly, errors were a cause of frustration for the students. Instead of applying a systematic debugging process using the poster, the students would quickly raise their hands to ask the teacher for help. These results can suggest that students did not consider the processes of pair programming and debugging as valuable towards their goal of an advanced result.

Paradoxically, one of the reasons pair programming was included in the design is that research suggests pair programming makes K-12 students write programs with fewer bugs (Campe et al., 2019). Similarly, the poster providing a process for systematic debugging was included in the design because research suggests systematic debugging increases the students' self-reliance in debugging (Michaeli & Romeike, 2019b). In other words, increased progression in the assignments was an expected outcome of including the two strategies. The students' focus on obtaining visually pleasing results rather than the programming process may have constrained their possibilities for obtaining the target knowledge in programming and trigonometry.

The discrepancy between the students' focus on the visual results and the designed intention to focus more on the process can indicate that the benefits of pair programming and systematic debugging were not communicated clearly enough to the students throughout the three sessions. Campe et al. (2019) emphasized in their pair programming toolkit that the teacher needs to scaffold the pair programming process by reminding the students about the value of pair programming, among others. Another possible explanation for the discrepancy is that the students did not learn the methods of pair programming and debugging sufficiently. Research on debugging education emphasizes how debugging is a complex skill (Li et al., 2019) that requires specific teaching lessons (Michaeli & Romeike, 2019a). Therefore, an increased emphasis on the benefits of pair programming and debugging prior to and throughout the lessons is a suggestion to improve the design. Furthermore, training the students in applying the techniques as prerequisites could be appropriate.

However, the student interviews show that the students' perceptions of pair programming were not solely dismissive. The students reflected on the advantages of pair programming in the interviews. One student highlighted the possibility of constructive feedback from their partner. Another student also felt that he understood programming better from having to explain his thoughts in simple words as a navigator. These reflections are in line with the benefits identified in research on pair programming Campe et al. (2019). These statements reinforce the suggestion to improve the training and scaffolding of pair programming in the didactical situation due to the potential for beneficial effects for obtaining the target knowledge.

8.4 Technical Overhead

In session 2, one of the student pairs produced a program with a logical error related to combining the Python libraries math and turtle in their program. The libraries respectively take in arguments with radians and degrees as angle measures, requiring conversions between the measures in the program. The issue was foresighted through the preliminary mathematical analysis as described in Chapter 4.1, and an attempt to counter the problem was made in the design. The Python Cheat Sheet provided the students with conversion examples between degrees and radians using the math library. However, another student pair carried out the conversion from degrees to radians correctly in session 2. These results indicate that combining the two Python libraries led to technical overhead for the students, but the hurdle is possible to overcome with provided examples. Regardless of the possibility of overcoming the hurdle, it was a constraining factor for the progression in the assignments, thus constraining the possibilities for obtaining the target knowledge in both programming and trigonometry. This result opens up to question the main rationale for teaching programming in this context. Is it for students to apply programming in solving mathematical problems, or is learning a widespread programming language like Python also important? According to Guzdial (2015), if the main rationale is to apply programming to mathematical problems, it is worth considering using special-purpose educational programming languages to limit the technical overhead. Consequently, the choice of TLEs in the didactical situation is worth considering, given that there is a possibility for autonomously selecting TLEs.

8.5 Misconceptions of Loops

One of the results from session 3 was that the student programs that did not reach the final assignment had various implementations of repeated function calls. This result can be seen in conjunction with two other results. Firstly, one result from session 2 was that several student programs included redundant for-loops performing one iteration of an instruction sequence. Secondly, only one of the programs delivered in session 3 included a while-loop. A proximate interpretation of these results is that the students did not have the expected understanding of repetition and the associated Python programming constructs of for-loops and while-loops. If the students had misconceptions about loops and repetition, they brought the misconceptions along when implementing repeated function calls. Therefore, an improvement of the didactical situation is to ensure that the students have a solid understanding of loops as a prerequisite.

8.6 Imprecise Wording in the Assignments

A different interpretation of the various implementations of repeated function calls is connected to the design of the assignments given to the students. As described in the preliminary analysis of subroutines, subroutines were invented to solve repeated calls to an instruction sequence with different arguments passed to solve a family of related problems. The assignments given to the students in session 3 never specified that the triangles drawn should be non-congruent. Consequently, the students did not have to isolate the functionality for drawing one triangle from the repetition to complete the assignment. This can be exemplified through one variation of repeated function calls, where repetition was included in the function definition, and the function was only called once in the program. Reformulating the wording in the assignments to include the requirement of drawing non-congruent triangles could facilitate the implementation of more generalized subroutines and isolate the function from repeated function calls.

8.7 Local Scope in Subroutines

One of the results from session 3 was that one of the student pairs working on the last assignment confused a global variable with a function parameter having the same name side_length. The students believed that the global variable was instantiated during the function declaration or the function call. This result is in line with the findings from Kallia and Sentance (2017), where parameter passing and variable scope are identified as two of 11 concepts that pose extra difficulties to secondary school students learning computer programming. The confusion among the students points back to a weakness in the epistemological analysis of subroutines, where the concept of local scope as an inherent characteristic of subroutines was not discovered. The weakness in the epistemological analysis was continued in the design, where the local scope was not explicitly addressed during the devolution phases in sessions 2 and 3. A subsequent suggestion to improve the design is to emphasize local scope in the didactical situation.

8.8 Unsystematic Debugging

The challenge of handling errors for programming novices is known in the research on computing education on the K-12 level (Li et al., 2019; Michaeli & Romeike, 2019b). A systematic debugging process embodied through a poster in the milieu was one of the measures in the design for aiding students' debugging process. However, the results indicate that the students chose not to use the debugging poster and that error handling was challenging for the students.

The students highlighted that handling logical errors were particularly difficult to resolve, leading to frustration and being stuck. Their strategies to handle errors were unsystematically tracing their program, drawing from previous programming experience, or asking the teacher for help. The observations of Dina and Vegard encountering a logical error in session 3 were in accordance with the students' perceptions of their strategies for handling errors. The students traced their

program, looking for syntax errors before asking the teacher for help.

From these results, it can be argued that the lack of systematic debugging among the students constrained their possibilities for obtaining the target knowledge both in mathematics and programming. Consequently, incorporating the teaching of a systematic debugging process is a potential improvement in the designed didactical situation. A concrete suggestion is to ensure systematic debugging as a student prerequisite. Aiding students in debugging logical errors is an impactful didactical variable when teaching programming with mathematics. The design choices did not sufficiently support the students debugging process. Consequently, the students did not follow the progression in the assignments.

8.9 Duration

The lectures had a total duration of $5 \cdot 45$ minutes. The time frame available was fixed by the teachers whose classes were visited in this study. However, the set of target knowledge for the lecture sequence was defined in this study and is open for discussion. The students' programs indicate that most student pairs did not obtain the target knowledge. It is pertinent to address whether the students were given adequate time to work on the assignments. According to Professor of Electrical Engineering and Computer Science Mark Guzdial, when combining the teaching of programming with another subject, the students require more time than when teaching just one subject separately (Guzdial, 2015). The time frame may have been a constraining factor for the target knowledge. It is relevant to either reduce the expectations or extend the time frame in future realizations of this lecture sequence. Especially providing more time for the adidactical work phase in session 3 could be relevant, as the last assignments seem to evoke the application of the target knowledge.

9 Conclusion

9.1 Answering the Research Questions

The new National Curriculum enrolled in 2020 in Norway led to the inclusion of programming in the existing mathematics subjects, among them Mathematics 1T (Norwegian Directorate for Education and Training, 2020b). Programming contained in the mathematics subject is a sparsely researched context for both mathematics and programming education (Forsström & Kaufmann, 2018; Waite, 2017). The new situation requires research to gain insight into the associated challenges and benefits. This study has addressed the problem by investigating the main research question:

> How can a designed didactical situation in Mathematics 1T affect students' opportunities for obtaining the target knowledge in programming and trigonometry?

The main research question was explored through four more specific research questions. The research questions are answered two and two together, as the results from this study have shown an entanglement between the contributing factors for obtaining the target knowledge in trigonometry and programming.

> **RQ1:** Which factors in the didactical situation contribute toward students' possibilities for obtaining the target knowledge in trigonometry?
> **RQ2:** Which factors in the didactical situation contribute toward students' possibilities for obtaining the target knowledge in programming?

The student programs show that a few of the student pairs applied The Law of Cosines and The SAS Theorem Area Formula, and they also applied their associated subroutine appropriately for repeatedly drawing non-congruent triangles. The result points to a fruitful synergy between the application of trigonometry and programming in the main problem given to the students. The synergy created was evoked in the last assignments given to the students, making the progression in the assignments a conditioning factor for their possibilities for obtaining both target knowledge.

Visual feedback and visually pleasing results contributed to student engagement and persistence. The visual feedback was enabled by including the Python turtle library in the milieu. The PRIMM structure of the assignments positively influenced engagement and persistence by providing visually pleasing results quickly. Furthermore, the PRIMM structure contributed to students' ownership of their products by gradually increasing the modification of the provided programs. The high level of engagement and persistence is likely to have aided the students' progression while working with the assignments and consequently contributed to their possibilities for obtaining the target knowledge.

Pair programming provided the students with easy access to help and constructive feedback from their partners. The navigator role also appears to have strengthened the students' understanding of the programming aspect of the content at hand. The results indicate that the benefits of pair programming did not fully come through to the students, calling for better implementation and scaffolding of the pair programming methodology to increase positive outcomes. Nevertheless, the pair programming methodology seems to have positively influenced the students' understanding of their programs and access to help and feedback. Thus, pair programming has to some extent, contributed to the students' possibilities for obtaining the target knowledge.

> **RQ3:** Which factors in the didactical situation hinder the students' possibilities for obtaining the target knowledge in trigonometry?
> **RQ4:** Which factors in the didactical situation hinder the students' possibilities for obtaining the target knowledge in programming?

A majority of the students delivered programs applying lower-level trigonometry than intended, such as the Pythagorean Theorem. Furthermore, their implementations of subroutines did not

reflect the intended characteristic of solving a family of related problems, as they drew congruent triangles. These findings reflect that the students were partially constrained from progressing to the last assignments and thus also constrained in their process towards obtaining the target knowledge.

One of the identified constraining factors was the students' focus on producing visually pleasing results rather than focusing on the process of getting there. The students relied on unsystematic debugging strategies and asked the teacher for help when encountering errors, leading to frustration and being stuck. They did not use the debugging poster provided to aid them in applying a systematic debugging process. The unsystematic debugging process hindered the students' progression in the assignments and their possibilities for obtaining the target knowledge.

Furthermore, the duration of the didactical situation constrained the students' progression. The total duration of all sessions was $5 \cdot 45$ minutes. The lack of time to progress further in the assignments, especially in session 3, constrained the students' possibilities for obtaining the target knowledge.

Relatedly, technical overhead was posed on the students due to conversion between degrees and radians, two angle measurement units. Including both the turtle and math libraries in the milieu imposed the conversions, as the libraries require arguments measured in degrees and radians, respectively. The conversions were technical overhead as they were required to implement but did not add value in regards to the target knowledge.

The results indicate two misconceptions about programming constructs constraining the students' possibilities for obtaining the target knowledge. Firstly, the students seemed not to have the expected prerequisites concerning loops. The misconception influenced the students' repeated function calls, resulting in more or less convenient implementations. The second misconception was related to the local scope for parameters in subroutines perceived as global by the students. The local scope characteristic was not addressed to the students, as it was not identified as an inherent difficulty of subroutines in the epistemological analysis. Consequently, a student pair encountered an error related to parameter scope they could not resolve.

Imprecise wording in the assignments also constrained the students' possibilities for obtaining the target knowledge. The assignments never specified that the students should draw non-congruent triangles. The imprecise wording likely influenced most student pairs to implement subroutines that did not solve a family of related problems but rather drew congruent triangles.

9.2 Implications

This study shows how a didactical situation can be designed to combine progression towards target knowledge in programming and trigonometry in the Mathematics 1T subject. The analysis of the classroom realizations has provided insight into contributing and constraining factors in the didactical situation that can be used to improve the design before future classroom realizations. Trigonometry and programming are also included in a subsequent mathematics subject, Mathematics R2, on the upper secondary level in Norway(Norwegian Directorate for Education and Training, 2021). The insights into constraining and contributing factors identified in this didactical situation could be used to inform the teaching of trigonometry and programming in Mathematics R2.

Inherent in the Didactical Engineering research methodology is the highly contextualized nature of the findings (Artigue, 2015). This study is situated in an early phase of the inclusion of programming into mathematics subjects in Norway, which affects the students' prerequisites. Several findings from this study addressed the students' lack of prerequisites regarding loops, pair programming, and debugging. The lack of prerequisites might not be problematic a few years from now, as programming has been included in primary and lower secondary mathematics subjects (Norwegian Directorate for Education and Training, 2020a). However, the identification of said prerequisites can be useful in deciding the order of concepts to teach students. In particular, teaching loops prior to subroutines seems appropriate by the results of this study. Furthermore, from the results, it can be suggested to focus early on processes such as pair programming and systematic debugging.

The preliminary analyses are not as contextualized as the results from the classroom realizations. The epistemological and didactical analyses can be developed further and applied in new research projects and contexts (Artigue, 2015). The local scope misconception tracing back to the weakness of the epistemological analysis is a finding from this study that is applicable in other settings where the application of subroutines is the intended target knowledge.

Lastly, this study has shown that the DE research methodology combined with the concepts provided by TDS can produce valuable insight close to the field of practice into teaching target knowledge in programming and trigonometry. The methodology and conceptual framework in this study can inspire future research projects combining intended target knowledge in programming and mathematics in Norway.

9.3 Limitations

There are several limitations to this research project worth addressing. This project was carried out over the course of one semester. Consequently, the time devoted to carrying out the preliminary analyses of the target knowledge was limited. As proven by the identified weakness in the epistemological analysis described earlier, the lack of time reduced the depth of the preliminary analyses. The study also has an interdisciplinary scope. The increased breadth of the study further limited the depth of the preliminary analyses.

Another limitation is related to the internal generalizability of the results from observations. One pair from each of the two classes were recorded in the adidactical work phases. As the quality of the audio recorded during the realizations in the first class was too poor, only the observational data from the second class could be transcribed and analyzed. Consequently, the observations of work processes rely on only one student pair working together. The sparse observational data material is a threat to the internal generalizability of the results, meaning that the results presented are not necessarily valid for the rest of the student pairs than the pair observed (Robson & McCartan, 2016).

It is also worth discussing the validity of the results from this study. The study conducted was associated with my master's thesis, meaning that I am the only researcher who transcribed and analyzed the data material. Similarly, the data collected was in Norwegian, whereas this thesis is written in English. I translated the excerpts from the data material included in this report. These processes are subject to interpretation (Robson & McCartan, 2016). Having another researcher analyze and translate parts of the data material and then evaluate the compliance with my analysis would have strengthened the validity of the results.

9.4 Further Research

This study has aimed to answer four research questions. Along the process, countless new questions have emerged. Three relevant directions of future research are addressed below.

9.4.1 Computational Thinking

This study has been concerned with the inclusion of programming in the Mathematics 1T subject. Problem-solving with computational thinking has also been included in the Mathematics 1T curriculum alongside programming (Norwegian Directorate for Education and Training, 2020b). Programming and computational thinking are tightly connected (Wing, 2006). Consequently, the processes in computational thinking such as decomposition, abstraction, and pattern recognition (Dong et al., 2019) were themes that repeatedly appeared at different points of this study. Computational thinking has not been analyzed or addressed in this report to reduce the scope. Consequently, a direction for future research is investigating the potential for obtaining target knowledge within computational thinking with this designed didactical situation.

9.4.2 Digital Art Production in the Mathematics Subject

One of the experimental design choices in this study is the art production aspect of the problems given to the students. Working towards visually pleasing results was a contributing factor to the students' engagement. However, a thorough investigation into the students' perceptions of art production was not conducted. To my knowledge, digital art production in mathematics subjects on the K-12 level in Norway has not been widely researched. Consequently, the effect of introducing digital art production in Mathematics 1T, and other mathematics subjects, is relevant to investigate explicitly.

9.4.3 Didactical Engineering

DE inspired the research methodology in this study. DE was and still is, developed as part of the French didactics tradition for research on traditional mathematics education (Barquero & Bosch, 2015). This study has provided valuable insight into the factors contributing to and constraining the students' possibilities for obtaining target knowledge, both in trigonometry and programming. It is pertinent to address the potential for applying DE in further research to investigate dual focus on target knowledge in programming and other topics in mathematics than trigonometry, both in Mathematics 1T and other subjects in the primary and secondary education and training in Norway.

9.5 Professional Relevance

This master's thesis marks five years of studies in Natural Science with Teacher Education. Teaching mathematics and computing in Norwegian lower and upper secondary schools is my future everyday life as a professional teacher. I am looking forward to further developing and testing the sequence of lectures researched in this study. The DE design methodology that I have learned and adapted through this study is even more impactful. The methodology is applicable for evidence-based design and testing of my teaching practice in the future. The rigorousness and thoroughness of this study can not be accomplished in hectic day-to-day. However, a simplified version of DE and a scientific attitude towards my teaching practice should be achievable in collaboration with teacher colleagues every year to continue my professional development.

This study has also contributed to my understanding of how real-world qualitative research is conducted in an educational setting. I have obtained an inside perspective. In particular, I have gained insight into quality characteristics, such as taking ethical considerations seriously throughout the project. Consequently, conducting this project has given me a better starting point for critically evaluating research in education. Evaluating research is significant when I update myself in the field, which is part of my future professional work.

Bibliography

Abdüsselam, M. S., Turan-Güntepe, E. & Durukan, Ü. G. (2022). Programming education in the frameworks of reverse engineering and theory of didactical situations. *Education and Information Technologies*, *2022*. https://doi.org/10.1007/s10639-021-10883-8

Artigue, M. (2015). Perspectives on design research: The case of didactical engineering. In A. Bikner-Ahsbahs, C. Knipping & N. Presmeg (Eds.), *Approaches to qualitative research in mathematics education: Examples of methodology and methods* (pp. 467–496). Springer. https://doi.org/10.1007/978-94-017-9181-6_17

Aschehoug. (n.d.). *Grunnkurs i programmering med Python*. Retrieved 6th June 2022, from https://aunivers.lokus.no/fagpakker/realfag/programmering/python/ressurser/grunnkurs-i-programmering-med-python

Barquero, B. & Bosch, M. (2015). Didactic engineering as a research methodology: From fundamental situations to study and research paths. In A. Watson & M. Ohtani (Eds.), *Task design in mathematics education* (pp. 249–272). Springer. https://doi.org/10.1007/978-3-319-09629-2_8

Beck, K. & Andres, C. (2004). *Extreme programming explained : Embrace change* (2nd ed.). Addison-Wesley.

Bocconi, S., Chioccariello, A. & Earp, J. (2018). *The Nordic approach to introducing computational thinking and programming in compulsory education*. Report prepared for the Nordic@BETT2018 Steering Group. https://doi.org/10.17471/54007

Bosch, A. M. V. (2021a). *Integrating programming into* Matematikk 1T. [Unpublished student research report]. Department of Computer Science, Norwegian University of Science and Technology. Included in Appendix M.

Bosch, A. M. V. (2021b). *Programmering i matematikk 1T: Hvordan kan vi støtte matematikklærerne?* [Programming in mathematics 1T: How can we support the mathematics teachers?]. [Unpublished student research report]. Department of Mathematical Sciences, Norwegian University of Science and Technology. Included in Appendix N.

Brousseau, G. (2002). *Theory of didactical situations in mathematics: Didactique des mathématiques, 1970–1990* (N. Balacheff, M. Cooper, R. Sutherland & V. Warfield, Eds. & Trans.). Kluwer. https://doi.org/10.1007/0-306-47211-2

Campe, S., Green, E. & Denner, J. (2019). *K-12 pair programming toolkit*. ETR. https://www.etr.org/default/assets/File/projects/Pair%20Programming%20Toolkit_FINAL%207_30_19.pdf

Daher, W., Baya'a, N. & Jaber, O. (2022). Understanding prospective teachers' task design considerations through the lens of the theory of didactical situations. *Mathematics, 10*(3), Article 417. https://doi.org/10.3390/math10030417

Dasgupta, S. (2014). *It began with Babbage : The genesis of computer science*. Oxford University Press.

Denner, J., Werner, L., Campe, S. & Ortiz, E. (2014). Pair programming: Under what conditions is it advantageous for middle school students? *Journal of Research on Technology in Education*, *46*(3), 277–296. https://doi.org/10.1080/15391523.2014.888272

Dong, Y., Catete, V., Jocius, R., Lytle, N., Barnes, T., Albert, J., Joshi, D., Robinson, R. & Andrews, A. (2019). PRADA: A practical model for integrating computational thinking in K-12 education. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, *2019*, 906–912. https://doi.org/10.1145/3287324.3287431

Durand-Guerrier, V., Meyer, A. & Modeste, S. (2019). Didactical issues at the interface of mathematics and computer science. In G. Hanna, D. A. Reid & M. de Villiers (Eds.), *Proof technology in mathematics research and teaching* (pp. 115–138). Springer. https://doi.org/10.1007/978-3-030-28483-1_6

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

Gueudet, G., Buteau, C., Mesa, V. & Misfeldt, M. (2014). Instrumental and documentational approaches: From technology use to documentation systems in university mathematics education. *Research in Mathematics Education*, *16*(2), 139–155. https://doi.org/10.1080/14794802.2014.918349

Guzdial, M. (2015). *Learner-centered design of computing education: Research on computing for everyone*. Morgan & Claypool.

Johnson, M. (n.d.). *SinSystem2*. Retrieved 30th March 2022, from https://openprocessing.org/sketch/1358647

Kallia, M. & Sentance, S. (2017). Computing teachers' perspectives on threshold concepts: Functions and procedural abstraction. *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*, *2017*, 15–24. https://doi.org/10.1145/3137065.3137085

Kaufmann, O. T. & Stenseth, B. (2021). Programming in mathematics education. *International Journal of Mathematical Education in Science and Technology*, *52*(7), 1029–1048. https://doi.org/10.1080/0020739X.2020.1736349

Li, C., Chan, E., Denny, P., Luxton-Reilly, A. & Tempero, E. (2019). Towards a framework for teaching debugging. *Proceedings of the Twenty-First Australasian Computing Education Conference*, *2019*, 79–86. https://doi.org/10.1145/3286960.3286970

Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J. & Szabo, C. (2018). Introductory programming: A systematic literature review. *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, *2018*, 55–106. https://doi.org/10.1145/3293881.3295779

Michaeli, T. & Romeike, R. (2019a). Current status and perspectives of debugging in the K12 classroom: A qualitative study. *2019 IEEE Global Engineering Education Conference*, *2019*, 1030–1038. https://doi.org/10.1109/EDUCON.2019.8725282

Michaeli, T. & Romeike, R. (2019b). Improving debugging skills in the classroom: The effects of teaching a systematic debugging process. *Proceedings of the 14th Workshop in Primary and Secondary Computing Education, 2019*, Article 15. https://doi.org/10.1145/3361721.3361724

Ministry of Education and Research. (2017). *Framtid, fornyelse og digitalisering: Digitaliseringsstrategi for grunnopplæringen 2017–2021* [Future, renewal, and digitalization: Digitalization strategy for the primary and secondary education and training 2017–2021]. https://www.regjeringen.no / contentassets / dc02a65c18a7464db394766247e5f5fc / kd_framtid_fornyelse_digitalisering_nett.pdf

Mitchell, J. C. (2003). *Concepts in programming languages.* Cambridge University Press.

Norwegian Directorate for Education and Training. (2020a). *Curriculum for mathematics year 1–10 (MAT01-05)*. Established as regulations. The National curriculum for the Knowledge Promotion 2020. https://data.udir.no/kl06/v201906/laereplaner-lk20/MAT01-05.pdf?lang=eng

Norwegian Directorate for Education and Training. (2020b). *Læreplan i matematikk fellesfag vg1 teoretisk (matematikk T)(MAT09-01)* [Mathematics subject curriculum for vg1 theoretical (mathematics T)]. Established as regulations. The National curriculum for the Knowledge Promotion 2020. https://data.udir.no/kl06/v201906/laereplaner-lk20/MAT09-01.pdf?lang=nob

Norwegian Directorate for Education and Training. (2021). *Læreplan i matematikk for realfag (matematikk R)(MAT03-02)* [Mathematics subject curriculum for natural science (mathematics R)]. Established as regulations. The National curriculum for the Knowledge Promotion 2020. https://data.udir.no/kl06/v201906/laereplaner-lk20/MAT03-02.pdf?lang=nob

NTNU. (n.d.-a). *RFEL3100 - Research Methods in Mathematics and Science Education*. Retrieved 6th June 2022, from https://www.ntnu.edu/studies/courses/RFEL3100#tab=omEmnet

NTNU. (n.d.-b). *TDT4501 - Computer Science, Specialization Project*. Retrieved 6th June 2022, from https://www.ntnu.edu/studies/courses/TDT4501#tab=omEmnet

Pilgrim, M. (2009). *Dive into python 3*. Apress. https://doi.org/10.1007/978-1-4302-2416-7

Postholm, M. B. & Jacobsen, D. I. (2020). *Forskningsmetode for masterstudenter i lærerutdanningen* [Research methods for master's students in teacher education]. Cappelen Damm.

Python Software Foundation. (n.d.-a). math — Mathematical functions. In *The Python Standard Library*. Retrieved May 27th, 2022, from https://docs.python.org/3/library/math.html.

Python Software Foundation. (n.d.-b). random — Generate pseudo-random numbers. In *The Python Standard Library*. Retrieved May 27th, 2022, from https://docs.python.org/3/library/random.html.

Python Software Foundation. (n.d.-c). turtle — Turtle graphics. In *The Python Standard Library*. Retrieved May 27th, 2022, from. https://docs.python.org/3/library/turtle.html

Robson, C. & McCartan, K. (2016). *Real world research : A resource for users of social research methods in applied settings* (4th ed.). Wiley.

Rossen, E. (2019, December 5). programmering - IT [programming - IT]. In Store norske leksikon. https://snl.no/programmering_-_IT

Rother, K. (2017). *Pro python best practices: Debugging, testing and maintenance*. Apress. https://doi.org/10.1007/978-1-4842-2241-6

Sanne, A., Berge, O., Bungum, B., Jørgensen, E. C., Kluge, A., Kristensen, T. E., Mørken, K. M., Svorkmo, A.-G. & Voll, L. O. (2016). *Teknologi og programmering for alle - En faggjennomgang med forslag til endringer i grunnopplæringen - august 2016* [Technology and programming for all - A subject review with proposals for changes in compulsory education - August 2016]. Norwegian Directorate for Education and Training. https://www.udir.no/globalassets/filer/tall-og-forskning/forskningsrapporter/teknologi-og-programmering-for-alle.pdf

Sentance, S. & Waite, J. (2017). PRIMM: Exploring pedagogical approaches for teaching text-based programming in school. *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*, *2017*, 113–114. https://doi.org/10.1145/3137065.3137084

Sentance, S., Waite, J. & Kallia, M. (2019a). Teachers' experiences of using PRIMM to teach programming in school. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, *2019*, 476–482. https://doi.org/10.1145/3287324.3287477

Sentance, S., Waite, J. & Kallia, M. (2019b). Teaching computer programming with PRIMM: A sociocultural perspective. *Computer Science Education*, *29*(2-3), 136–176. https://doi.org/10.1080/08993408.2019.1608781

Sevik, K. (2016). *Programmering i skolen* [Programming in school]. Senter for IKT i utdanningen. https://www.udir.no/globalassets/filer/programmering_i_skolen.pdf

Stadsvoll, V. (2020). *Et design av en didaktisk situasjon for introduksjonen av trigonometri etter prinsippene i didaktisk ingeniørvirksomhet* [A design of a didactical situation for the introduction of trigonometry after the principles in Didactic Engineering] [Master's thesis, Norwegian University of Science and Technology]. NTNU Open. https://hdl.handle.net/11250/2778352

Strømskag, H. (2020). Teorien for didaktiske situasjoner i matematikk [The theory of didactical situations in mathematics]. In V. Nilssen & S.-M. Høynes (Eds.), *Samtaleorientert matematikk* (pp. 23–80). Fagbokforlaget.

The National Committee for Research Ethics in the Social Sciences and the Humanities. (2019). *Guidelines for research ethics in the social sciences, humanities, law and theology*. https://www.forskningsetikk.no/en/guidelines/social-sciences-humanities-law-and-theology/guidelines-for-research-ethics-in-the-social-sciences-humanities-law-and-theology/

Tjora, A. H. (2021). *Kvalitative forskningsmetoder i praksis* [Qualitative research methods in practice] (4th ed.). Gyldendal.

University of Oslo. (2021, May 18). *Nettskjema-dictaphone*. https://www.uio.no/english/services/it/adm-services/nettskjema/help/nettskjema-dictaphone.html

Venema, G. A. (2012). *Foundations of geometry* (2nd ed.). Pearson.

Waite, J. (2017). *Pedagogy in teaching computer science in schools: A literature review*. Retrieved March 30th, 2022, from https://royalsociety.org/computing-education.

Weisstein, E. (n.d.-a). 30-60-90 Triangle. In *MathWorld – A Wolfram Web Resource*. Retrieved May 28th, 2022, from https://mathworld.wolfram.com/30-60-90Triangle.html.

Weisstein, E. (n.d.-b). Geometric Congruence. In *MathWorld – A Wolfram Web Resource*. Retrieved May 13th, 2022, from https://mathworld.wolfram.com/GeometricCongruence.html.

Weisstein, E. (n.d.-c). Injection. In *MathWorld – A Wolfram Web Resource*. Retrieved May 13th, 2022, from https://mathworld.wolfram.com/Injection.html.

Weisstein, E. (n.d.-d). Law of Cosines. In *MathWorld – A Wolfram Web Resource*. Retrieved May 10th, 2022, from https://mathworld.wolfram.com/LawofCosines.html.

Weisstein, E. (n.d.-e). Law of Sines. In *MathWorld – A Wolfram Web Resource*. Retrieved May 10th, 2022, from https://mathworld.wolfram.com/LawofSines.html.

Weisstein, E. (n.d.-f). SAS Theorem. In *MathWorld – A Wolfram Web Resource*. Retrieved May 10th, 2022, from https://mathworld.wolfram.com/SASTheorem.html.

Weisstein, E. (n.d.-g). Triangle. In *MathWorld – A Wolfram Web Resource*. Retrieved May 10th, 2022, from https://mathworld.wolfram.com/Triangle.html.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

A NSD Approval

Notification form / Programmering i matematikk 1T / Assessment

Assessment

Reference number
826023

Project title
Programmering i matematikk 1T

Data controller (institution responsible for the project)
Norges teknisk-naturvitenskapelige universitet / Fakultet for informasjonsteknologi og elektroteknikk (IE) / Institutt for datateknologi og informatikk

Project leader
Majid Rouhani

Student
Anne Margrethe Vestgøte Bosch

Project period
10.01.2022 - 01.10.2022

Notification Form

Date
24.02.2022

Type
Standard

Comment
Det er vår vurdering at behandlingen av personopplysninger i prosjektet vil være i samsvar med personvernlovgivningen så fremt den gjennomføres i tråd med det som er dokumentert i meldeskjemaet med vedlegg, og eventuelt i meldingsdialogen mellom innmelder og Personverntjenester. Behandlingen kan starte.

TYPE OPPLYSNINGER OG VARIGHET
Prosjektet vil behandle alminnelige kategorier av personopplysninger frem til den datoen som er oppgitt i meldeskjemaet.

LOVLIG GRUNNLAG
Prosjektet vil innhente samtykke fra de registrerte til behandlingen av personopplysninger. Vår vurdering er at prosjektet legger opp til et samtykke i samsvar med kravene i art. 4 og 7, ved at det er en frivillig, spesifikk, informert og utvetydig bekreftelse som kan dokumenteres, og som den registrerte kan trekke tilbake.

Lovlig grunnlag for behandlingen vil dermed være den registrertes samtykke, jf. personvernforordningen art. 6 nr. 1 bokstav a.

PERSONVERNPRINSIPPER
Personverntjenester vurderer at den planlagte behandlingen av personopplysninger vil følge prinsippene i personvernforordningen om:
· lovlighet, rettferdighet og åpenhet (art. 5.1 a), ved at de registrerte får tilfredsstillende informasjon om og samtykker til behandlingen
· formålsbegrensning (art. 5.1 b), ved at personopplysninger samles inn for spesifikke, uttrykkelig angitte og berettigede formål, og ikke behandles til nye, uforenlige formål
· dataminimering (art. 5.1 c), ved at det kun behandles opplysninger som er adekvate, relevante og nødvendige for formålet med prosjektet
· lagringsbegrensning (art. 5.1 e), ved at personopplysningene ikke lagres lengre enn nødvendig for å oppfylle formålet

DE REGISTRERTES RETTIGHETER
Så lenge de registrerte kan identifiseres i datamaterialet vil de ha følgende rettigheter: innsyn (art. 15), retting (art. 16), sletting (art. 17), begrensning (art. 18), og dataportabilitet (art. 20).

Personverntjenester vurderer at informasjonen om behandlingen som de registrerte vil motta oppfyller lovens krav til form og innhold, jf. art. 12.1 og art. 13.

Vi minner om at hvis en registrert tar kontakt om sine rettigheter, har behandlingsansvarlig institusjon plikt til å svare innen en måned.

FØLG DIN INSTITUSJONS RETNINGSLINJER
Personverntjenester legger til grunn at behandlingen oppfyller kravene i personvernforordningen om riktighet (art. 5.1 d), integritet og konfidensialitet (art. 5.1. f) og sikkerhet (art. 32).

Panopto er databehandler i prosjektet. Personverntjenester legger til grunn at behandlingen oppfyller kravene til bruk av databehandler, jf. art 28 og 29.

For å forsikre dere om at kravene oppfylles, må dere følge interne retningslinjer og/eller rådføre dere med behandlingsansvarlig institusjon.

MELD VESENTLIGE ENDRINGER
Dersom det skjer vesentlige endringer i behandlingen av personopplysninger, kan det være nødvendig å melde dette til oss ved å oppdatere meldeskjemaet. Før du melder inn en endring, oppfordrer vi deg til å lese om hvilke type endringer det er nødvendig å melde: https://www.nsd.no/personverntjenester/fylle-ut-meldeskjema-for-personopplysninger/melde-endringer-i-meldeskjema

Du må vente på svar fra oss før endringen gjennomføres.

OPPFØLGING AV PROSJEKTET
Personverntjenester vil følge opp ved planlagt avslutning for å avklare om behandlingen av personopplysningene er avsluttet.
Lykke til med prosjektet!

Assessment

Reference number
826023

Project title
Programmering i matematikk 1T

Data controller (institution responsible for the project)
Norges teknisk-naturvitenskapelige universitet / Fakultet for informasjonsteknologi og elektroteknikk (IE) / Institutt for datateknologi og informatikk

Project leader
Majid Rouhani

Student
Anne Margrethe Vestgøte Bosch

Project period
10.01.2022 - 01.10.2022

Notification Form

Date
04.03.2022

Type
Standard

Comment
Personverntjenester har vurdert endringen registrert i meldeskjemaet.

Det er vår vurdering at behandlingen av personopplysninger i prosjektet vil være i samsvar med personvernlovgivningen så fremt den gjennomføres i tråd med det som er dokumentert i meldeskjemaet med vedlegg. Behandlingen kan fortsette.

Endringen består av at Nettskjema er lagt til som databehandler i prosjektet. Vi legger til grunn at behandlingen oppfyller kravene til bruk av databehandler, jf. art 28 og 29.

OPPFØLGING AV PROSJEKTET

Vi vil følge opp underveis ved planlagt avslutning for å avklare om behandlingen av personopplysningene er avsluttet.

Kontaktperson: Henning Levold
Lykke til videre med prosjektet!

B Interview Guide in Norwegian

Intervjuguide

Spørsmålene i dette intervjuet vil kun være av didaktisk natur, og vil kun gå på observasjoner/bemerkninger/forståelse/oppklaringer rundt intervjuobjektenes tidligere skriftlige og praktiske besvarelser. Ingen personlige opplysninger vil bli nevnt overhodet. Jeg vil komme til å stille spørsmål til intervjuobjektene knyttet til skriftlig og/eller praktisk besvarelse, og de svarer på spørsmålene etter beste evne. Jeg vil også stille spørsmål for å få en dypere forståelse av intervjuobjektenes motivasjon i faget. Samtalen vil vare omtrent 30 minutter, og vil finne sted for å få en dypere forståelse av hvorfor intervjuobjektene har svart og gjort det de har gjort. Da intervjuet er semistrukturert vil det kunne ta ulike retninger baser på hva som dukker opp underveis.

Eksempler på spørsmål vil være:

Hvorfor har du/dere valgt å bruke en <funksjon, while-løkke…> her?

Hvorfor har du/dere kalt denne variabelen for <...>?

Hva synes du/dere om å jobbe med programmering i matematikk, kontra «vanlig» undervisning?

Hva synes du/dere var vanskelig med oppgavene? Hvorfor?

Hva likte eller mislikte du/dere med oppgavene? Hvorfor?

Hva har du lært ved å bruke programmering i Trigonometri?

C Form of Consent

Anne Margrethe Vestgøte Bosch
<adresse>
<tlf.>
<epost>

<Sted>, 7. februar 2022

Til elever i Matematikk 1T ved [redacted] videregående skole

Anmodning om tillatelse til innsamling av elevbesvarelser og lydopptak av intervju, samt videoopptak og skjermopptak.

Mitt navn er Anne, og jeg er student på 5. året ved lektorprogrammet i realfag på NTNU. Nå i vår skal jeg skrive min masteroppgave om programmering i matematikk, og jeg ønsker derfor å besøke klassen deres for å samle inn forskningsdata til prosjektet. Prosjektet går ut på å finne ut hvordan man kan bruke programmering i trigonometri på en motiverende måte, og hvilke virkninger det kan ha for læringen i faget Matematikk 1T. Klassen skal jobbe i grupper med programmeringsoppgaver knyttet til trigonometri i matematikktimene en uke. Jeg vil samle inn programmene som elevene lager, og i tillegg ta videoopptak av en (frivillig) gruppe, for å analysere hvordan de har arbeidet med oppgavene. Selv om bare en av gruppene filmes, vil lyd fra andre grupper i klasserommet kunne bli fanget opp. Jeg vil også ta opptak av PC-skjermen til gruppa mens de arbeider for å se utviklingen av programmet underveis.

Jeg vil i etterkant ta kontakt med 1-3 av elevgruppene for et lite intervju på ca. 30 minutter, med lydopptak. Under samtalen kommer vi til å diskutere hva elevene har gjort, og hvordan de har tenkt for å komme frem til svarene sine. Samtalen kommer derfor kun til å komme inn på generelle spørsmål knyttet til elevenes besvarelser, og forståelse rundt oppgavene som har blitt besvart. Ingen personlige opplysninger vil komme til å bli nevnt under intervjuet. Det eneste jeg trenger av personlige opplysninger fra elevene, er navn på programmene de har laget, slik at jeg kan finne dem igjen til et eventuelt intervju. Navnene vil selvfølgelig bli anonymisert når masteroppgaven min skal skrives. Alle andre personopplysninger vil også bli fullstendig anonymisert etter innsamlingen av videomaterialet.

Jeg trenger navn på elevene for å samtykke til innsamling av data, derunder et eventuelt intervju med lydopptak og/eller videoopptak av arbeidet. Jeg trenger også navn på programmene for å finne igjen de elevene jeg ønsker å snakke med etter den skriftlige datainnsamlingen. Under intervjuet har jeg behov for lydopptak for å få så godt dokumenterte data som mulig. Med en mest mulig nøyaktig gjengivelse av det som blir sagt, kan jeg igjen danne et bedre bilde av elevenes faglige utbytte. Siden jeg jobber alene, er det derfor mest hensiktsmessig å kunne høre besvarelsene i sin helhet i etterkant. Det samme gjelder også videoopptakene og skjermopptakene.

Jeg ber derfor om tillatelse til å samle inn materiale produsert av elevene, samt til å kunne ta videoopptak/skjermopptak av arbeidet, og lydopptak av samtalene i etterkant. Forutsetningen for tillatelsen er at alt innsamlet materiale blir behandlet med respekt og blir fullstendig

anonymisert, og at prosjektet ellers følger gjeldende retningslinjer for etikk og personvern. **Det er selvfølgelig HELT frivillig å delta, og man kan til enhver tid trekke seg fra deltakelse UTEN å måtte oppgi noen grunn til det.** De som ikke ønsker å delta i prosjektet vil gjennomføre den samme undervisningen, men data vil ikke samles inn om dem. Hvis du ikke vil delta, trenger du heller ikke å levere inn svarslippen på neste side.

Materialet som blir samlet inn vil kun bli sett og hørt av meg, og eventuelt av min masterveileder ved NTNU. Når prosjektet presenteres vil involverte personer bli anonymisert. Lyd- og videoopptakene vil bli tatt opp med opptakere fra NTNUs eiendom, og vil bli lagret på NTNU sin lagringstjeneste «Hjemmeområdet». Skjermopptakene vil bli lagret gjennom programvaren Panopto. Alle innsamlede data vil bli slettet etter at prosjektet er avsluttet, og senest 1.oktober 2022.

Så lenge du kan identifiseres i datamaterialet, har du rett til:

- innsyn i hvilke opplysninger vi behandler om deg, og å få utlevert en kopi av opplysningene
- å få rettet opplysninger om deg som er feil eller misvisende
- å få slettet personopplysninger om deg
- å sende klage til Datatilsynet om behandlingen av dine personopplysninger

Hvis du/dere vil vite mer om dette, eller hva det innsamlede materialet skal brukes til, så er det bare å ta kontakt med meg når som helst på telefon eller epost (se øverst i dokumentet for detaljer).

Faglig ansvarlig ved NTNU er Majid Rouhani: tlf.: 735 59 355; epost: majid.rouhani@ntnu.no

NTNUs personvernombud er Thomas Helgesen: tlf. +47 930 79 038; epost thomas.helgesen@ntnu.no.

Hvis du/dere har spørsmål knyttet til NSD sin vurdering av prosjektet, ta kontakt med:

- NSD – Norsk senter for forskningsdata AS på epost (personverntjenester@nsd.no) eller på telefon: 55 58 21 17.

Jeg håper du synes forskningen min er av verdi, og at du er villig til å være med på den. Jeg ber om at svarslippen på neste side fylles hvis du gir tillatelse til deltakelse i prosjektet.

På forhånd takk!

Vennlig hilsen
Anne M. V. Bosch

Tillatelse

Som del av mitt masterprosjekt ved lektorutdanningen i realfag ved NTNU ber jeg om din tillatelse til innsamling av digital besvarelse av programmeringsoppgaver og videoopptak i klasserommet, samt en eventuell samtale med lydopptaker, og et eventuelt skjermopptak av PC. Jeg ber også om tillatelse til å kopiere/bruke besvarelsen(e) som du har produsert til å skrive en fagartikkel rundt elevers forståelse av trigonometri.

Forutsetningen for tillatelsen er at besvarelser og annet innsamlet materiale blir fullstendig **anonymisert** og behandlet med **respekt**, og at prosjektet følger gjeldende retningslinjer for **etikk** og **personvern**. Samtykke kan trekkes tilbake når som helst, og uten begrunnelse.

☐ Jeg (eleven) gir tillatelse

Dato: ……

Elevens fornavn og etternavn: …….…….…….....

Underskrift av eleven:

…….....

Vennligst returner svarslippen til læreren din så snart som mulig.

D Form of Consent (Pilot)

Anne Margrethe Vestgøte Bosch
<Adresse>
<tlf.>
<epost>

<Sted>, 7. februar 2022

Vil du delta i en pilotstudie for forskningsprosjektet «Programmering i matematikk 1T»?

Anmodning om tillatelse til innsamling av studentprogrammer og lydopptak av intervju, samt videoopptak og skjermopptak.

Mitt navn er Anne, og jeg er student på 5. året ved lektorprogrammet i realfag på NTNU. Nå i vår skal jeg skrive min masteroppgave om programmering i matematikk, og jeg ønsker derfor å gjennomføre en workshop for lektorstudenter som en pilot før gjennomføring av utprøving i klasserommet. Prosjektet går ut på å finne ut hvordan man kan bruke programmering i trigonometri på en motiverende måte, og hvilke virkninger det kan ha for læringen i faget Matematikk 1T. Deltakerne skal jobbe i grupper med programmeringsoppgaver knyttet til trigonometri i én til to økter. Jeg vil samle inn programmene som studentene lager, og i tillegg ta videoopptak av en til to (frivillige) grupper, for å analysere hvordan de har arbeidet med oppgavene. Selv om bare to av gruppene filmes, vil lyd fra andre grupper i rommet kunne bli fanget opp. Jeg vil også ta opptak av PC-skjermene til de to gruppene mens de arbeider for å se utviklingen av programmet underveis.

Jeg vil i etterkant ta kontakt med 1-3 av gruppene for et lite intervju på ca. 30 minutter, med lydopptak. Under samtalen kommer vi til å diskutere hva studentene har gjort, og hvordan de har tenkt for å komme frem til svarene sine. Samtalen kommer derfor kun til å komme inn på generelle spørsmål knyttet til studentenes besvarelser, og forståelse rundt oppgavene som har blitt besvart. Ingen personlige opplysninger vil komme til å bli nevnt under intervjuet. Det eneste jeg trenger av personlige opplysninger er navn på programmene som er laget, slik at jeg kan finne dem igjen til et eventuelt intervju. Navnene vil selvfølgelig bli anonymisert når masteroppgaven min skal skrives. Alle andre personopplysninger vil også bli fullstendig anonymisert etter innsamlingen av videomaterialet.

Jeg trenger navn på studentene for å samtykke til innsamling av data, derunder et eventuelt intervju med lydopptak og/eller videoopptak av arbeidet. Jeg trenger også navn på programmene for å finne igjen de studentene jeg ønsker å snakke med etter den skriftlige datainnsamlingen. Under intervjuet har jeg behov for lydopptak for å få så godt dokumenterte data som mulig. Med en mest mulig nøyaktig gjengivelse av det som blir sagt, kan jeg igjen danne et bedre bilde av det faglige utbyttet. Siden jeg jobber alene, er det derfor mest hensiktsmessig å kunne høre besvarelsene i sin helhet i etterkant. Det samme gjelder også de to videoopptakene og skjermopptakene.

Jeg ber derfor om tillatelse til å samle inn skriftlig materiale produsert av studentene, samt til å kunne ta videoopptak/skjermopptak av arbeidet, og lydopptak av samtalene i etterkant. Det er da snakk om oppgavene som studentene har besvart først knyttet til programmering i matematikk, i tillegg til lydopptak av 1-3 intervjuer og to videoer av arbeid. Forutsetningen for tillatelsen er at alt innsamlet materiale blir behandlet med respekt og blir fullstendig anonymisert, og at prosjektet ellers følger gjeldende retningslinjer for etikk og personvern. **Det er selvfølgelig HELT frivillig å delta, og man kan til enhver tid trekke seg fra deltakelse UTEN å måtte oppgi noen grunn til det.** Hvis du ikke vil delta, trenger du heller ikke å levere inn svarslippen på neste side.

Materialet som blir samlet inn vil kun bli sett og hørt av meg, og eventuelt av min masterveileder ved NTNU. Når prosjektet presenteres vil involverte personer bli anonymisert. Lyd- og videoopptakene vil bli tatt opp med opptakere fra NTNUs eiendom, og vil bli lagret på NTNU sin lagringstjeneste «Hjemmeområdet». Skjermopptakene vil bli lagret gjennom programvaren Panopto. Alle innsamlede data vil bli slettet etter at prosjektet er avsluttet, og senest 1.oktober 2022.

Så lenge du kan identifiseres i datamaterialet, har du rett til:

- innsyn i hvilke opplysninger vi behandler om deg, og å få utlevert en kopi av opplysningene
- å få rettet opplysninger om deg som er feil eller misvisende
- å få slettet personopplysninger om deg
- å sende klage til Datatilsynet om behandlingen av dine personopplysninger

Hvis du/dere vil vite mer om dette, eller hva det innsamlede materialet skal brukes til, så er det bare å ta kontakt med meg når som helst på telefon eller epost (se øverst i dokumentet for detaljer).

Faglig ansvarlig ved NTNU er Majid Rouhani: tlf.: 735 59 355; epost: majid.rouhani@ntnu.no

NTNUs personvernombud er Thomas Helgesen: tlf. +47 930 79 038; epost thomas.helgesen@ntnu.no.

Hvis du/dere har spørsmål knyttet til NSD sin vurdering av prosjektet, ta kontakt med:

- NSD – Norsk senter for forskningsdata AS på epost (personverntjenester@nsd.no) eller på telefon: 55 58 21 17.

Jeg håper du synes forskningen min er av verdi, og at du er villig til å være med på den. Jeg ber om at svarslippen på neste side fylles hvis du gir tillatelse til deltakelse i prosjektet.

På forhånd takk!

Vennlig hilsen
Anne M. V. Bosch

Tillatelse

Som del av mitt masterprosjekt ved lektorutdanningen i realfag ved NTNU ber jeg om din tillatelse til innsamling av digital besvarelse av programmeringsoppgaver og videoopptak i klasserommet, samt en eventuell samtale med lydopptaker, og et eventuelt skjermopptak av PC. Jeg ber også om tillatelse til å kopiere/bruke besvarelsen(e) som du har produsert til å skrive en fagartikkel rundt elevers forståelse av trigonometri.

Forutsetningen for tillatelsen er at besvarelser og annet innsamlet materiale blir fullstendig **anonymisert** og behandlet med **respekt**, og at prosjektet følger gjeldende retningslinjer for **etikk** og **personvern**. Samtykke kan trekkes tilbake når som helst, og uten begrunnelse.

☐ Jeg (studenten) gir tillatelse

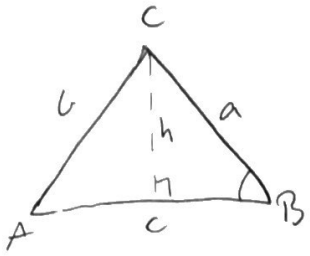
Dato: ……

Fornavn og etternavn: …….…….…….....

Underskrift:

…….....

E Area Formula Derived from The SAS Theorem

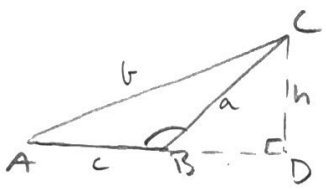


Acute triangle

I) $T = \frac{1}{2} \cdot c \cdot h$ Area of a triangle

II) $\frac{h}{a} = \sin B$ by definition

$T = \frac{1}{2} \cdot c \cdot a \cdot \sin B$ using II , q.e.d



Obtuse triangle

I) $T = \frac{1}{2} c \cdot h$ Area of a triangle

II) $\frac{h}{a} = \sin \angle CBD$ by definition

supplement angles

III) $\sin \angle CBD = \sin(180° - \angle CBD) = \sin \angle ABC = \sin B$ ←

IV $\frac{h}{a} = \sin B$ using II and III

$T = \frac{1}{2} \cdot c \cdot a \cdot \sin B$ using IV , q.e.d

F Proof of The Law of Sines

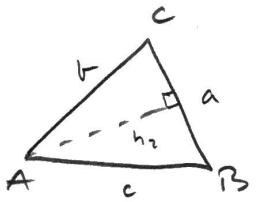
I)

Acute triangle

$$\frac{h_1}{b} = \sin A \quad \text{and} \quad \frac{h_1}{a} = \sin B \iff h_1 = \sin B \cdot a$$

$$\frac{h_1}{b} = \frac{a \cdot \sin B}{b} = \sin A \iff \frac{\sin B}{b} = \frac{\sin A}{a}$$

II)

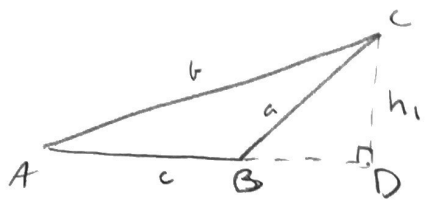


$$\frac{h_2}{b} = \sin C \quad \text{and} \quad \frac{h_2}{c} = \sin B \iff h_2 = \sin B \cdot c$$

$$\frac{h_2}{b} = \frac{\sin B \cdot c}{b} = \sin C \iff \frac{\sin B}{b} = \frac{\sin C}{c}$$

III) $$\frac{\sin A}{a} = \frac{\sin B}{b} = \frac{\sin C}{c} \iff \frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C}$$, q.e.d.

I) Obtuse triangle



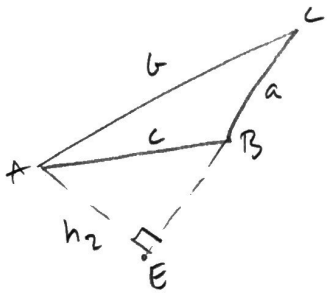
$$\frac{h_1}{b} = \sin A \quad \text{and} \quad \frac{h_1}{a} = \sin \angle CBD$$

where $\sin \angle CBD = \sin(180^\circ - \angle CBD) = \sin \angle ABC$

$$\Rightarrow \frac{h_1}{a} = \sin \angle CBD = \sin \angle ABC = \sin B$$

$$\frac{h_1}{b} = \frac{a \sin B}{b} = \sin A \iff \frac{\sin B}{b} = \frac{\sin A}{a}$$

II)



$$\frac{h_2}{b} = \sin C \quad \text{and}$$

$$\frac{h_2}{c} = \sin \angle ABE \quad \text{where}$$

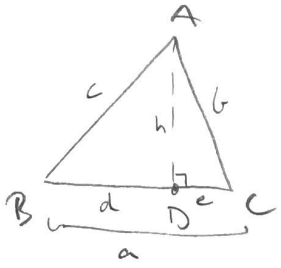
$$\sin \angle ABE = \sin(180^\circ - \angle ABE) = \sin \angle ABC$$

$$\Rightarrow \frac{h_2}{c} = \sin \angle CBD = \sin \angle ABC = \sin B$$

$$\frac{h_2}{b} = \frac{c \cdot \sin B}{b} = \sin C \iff \frac{\sin B}{b} = \frac{\sin C}{c}$$

III) $$\frac{\sin A}{a} = \frac{\sin B}{b} = \frac{\sin C}{c} \iff \frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C}$$ l.q.e.d.

G Proof of The Law of Cosines



Acute triangle

I $c^2 = d^2 + h^2$ Using the Pythagorean Theorem

II $b^2 = e^2 + h^2$

III $\cos C = \frac{e}{b}$ by definition

IV $d^2 = (a-e)^2$

$c^2 = d^2 + h^2$

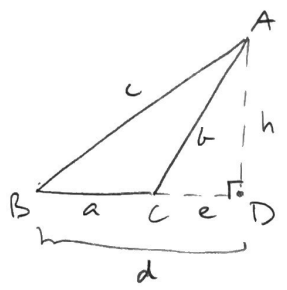
$c^2 = d^2 + b^2 - e^2$ Using II

$c^2 = (d-e)(d+e) + b^2$

$c^2 = (a-2e)a + b^2$ Using IV

$c^2 = a^2 + b^2 - 2ae$

$c^2 = a^2 + b^2 - 2a \cdot b \cdot \cos C$ Using III , q.e.d.



Obtuse triangle

I $c^2 = d^2 + h^2$ Using the Pythagorean Theorem

II $b^2 = e^2 + h^2$

III $\cos\angle ACD = \frac{e}{b}$ by definition

$\cos C = \cos\angle ACB$

$= \cos(180° - \angle ACD)$ Supplementary angles

$= -\cos\angle ACD$ Unit circle def.

IV $\cos C = -\frac{e}{b}$ Using III

V $d = a + e$

$c^2 = d^2 + h^2$

$c^2 = (a+e)^2 + h^2$ Using V

$c^2 = a^2 + 2ae + e^2 + h^2$

$c^2 = a^2 + b^2 + 2ae$ Using II

$c^2 = a^2 + b^2 - 2ab\cdot\cos C$ Using IV , q.e.d.

H Debugging Poster in Norwegian

I Python Cheat Sheet in Norwegian

Innebygde funksjoner i Python

Instruksjon	Beskrivelse
round(tall) round(tall, ndesimaler)	Runder av verdien til tall. round(10.555) → 11 round(10.555,2) → 10.56
int()	Gjør om en tekststreng (eller et desimaltall) til et heltall. Fjerner bare desimalene i desimaltall.
float("3.14")	Gjør om en tekststreng (eller et heltall) til et desimaltall.
input("Skriv inn et tall: ")	Skriver ut en tekst til brukeren og venter på svar. Svaret er alltid en tekststreng.
range(a,b) range(b)	Alle heltall fra og med a til b (men b er ikke med). range(1,10) → 1 til 9 range(5) → 0 til 4
print(a)	Skriver ut verdien til a

turtle-biblioteket

Flere instruksjoner finner du i dokumentasjonen. Du finner dokumentasjonen ved å søke i Google etter: python turtle documentation.

Instruksjon	Beskrivelse
import turtle	Importerer turtle-biblioteket til fila.
turtle.done() turtle.bye()	Begge linjene må legges inn nederst i fila for å hindre turtle-vinduet fra å krasje når programmet er ferdig og stoppe kjøringen av programmet når vi lukker turtle-vinduet.
turtle.shape("turtle")	Gjør om figuren som tegner til en skilpadde. Alternativer er "arrow", "turtle", "circle", "square", "triangle" og "classic".
turtle.pensize(2)	Angir strekens tykkelse. Standardverdien er 1.
turtle.color("magenta")	Angir strekens farge. Søk etter CSS colors på Google for å finne de mulige fargenavnene.
turtle.bgcolor("aliceblue")	Angir bakgrunnsfargen til tegnevinduet. Bruker de samme fargenavnene som i color().
turtle.fillcolor("cyan")	Angir fyllfarge. Bruker de samme fargenavnene som i color().
begin_fill() og end_fill()	Brukes før og etter en form tegnes for å fylle den med fargen som er angitt i fillcolor().
turtle.penup() og turtle.pendown()	Lar oss flytte figuren uten å tegne en strek. Bruk penup() for å slutte å tegne og pendown() for å begynne å tegne.
turtle.goto(x,y)	Skilpadden går til koordinatet (x , y). Origo er i midten av tegneområdet.
turtle.speed(1)	Angir tegnehastighet. Verdier fra 0 til og med 10 kan brukes. Økende verdi gir økende hastighet, med unntak av at 0 gir aller høyest hastighet.
turtle.clear()	Visker ut alt som er tegnet.

math-biblioteket

Instruksjon	Beskrivelse
import math	Importerer math-biblioteket til fila.
math.sqrt(a)	Finner kvadratroten av a.
vinkel_i_radianer = math.radians(v) sin_v = math.sin(vinkel_i_radianer)	math-biblioteket bruker ikke vinkelmål oppgitt i grader, men radianer. Vi gjør først om vinkelen v fra grader til radianer. Deretter finner vi verdien til sinus av vinkel v, og lagrer i variabelen sin_v.
vinkel_i_radianer = math.radians(v) cos_v = math.sin(vinkel_i_radianer)	Lagrer verdien til cosinus av vinkel v i variabelen cos_v.
sin_theta = a/c theta = math.degrees(math.asin(sin_theta))	Finner størrelsen til vinkelen theta når vi kjenner sinus til vinkelen. Vi må gjøre om fra radianer til grader.
cos_theta = b/c theta = math.degrees(math.acos(sin_theta))	Finner størrelsen til vinkelen theta når vi kjenner cosinus til vinkelen.

random-biblioteket

Instruksjon	Beskrivelse
import random	Importerer random-biblioteket til fila.
random.randrange(a,b)	Returnerer et tilfeldig heltall fra og med a, til b og med b-1.
random.uniform(a,b)	Returnerer et tilfeldig desimaltall fra og med a til og med b.

J Assignments in Norwegian

Økt 1

1. Les programmet på skjermen. Hva tror dere skjer når vi kjører programmet? Diskuter i parene og skriv ned en hypotese i skriveboka.
2. Bli enige om hvem som starter som sjåfør og kartleser. Sjåføren åpner Spyder på sin egen PC. Lag en ny fil som heter mystisk1.py og lagre den på skrivebordet. Skriv av programmet på skjermen. Kjør programmet.
3. Hva skjedde når dere kjørte koden? Hadde dere riktig i hypotesen deres?
   Hvis ikke, gjør om programmet slik at det gjør det som dere trodde. Skriv en kommentar i koden hvilken endring dere måtte gjøre.
4. Dere skal nå endre på programmet. Lag et nytt mål for hva programmet skal gjøre. Her er det lov å være kreativ! For eksempel: Programmet skal tegne 100 blå åttekanter.
   a. Skriv ned en plan steg for steg hva programmet skal gjøre for å nå målet. Kommentarer i koden kan brukes for å skrive planer.
   b. Endre på koden slik at den utfører planen dere lagde.
   c. Test om programmet virker slik som dere hadde tenkt.

Innsending:

a. Legg inn en kommentar med navnene deres i programmet og lagre.
b. Gi et nytt navn til fila slik at navnet passer til det som programmet gjør. For eksempel: blaa_aattekanter.py
c. Sjåføren leverer inn programmet på minnepenn.
d. Kartleseren leverer inn arkene på kateteret.

Økt 2:

Del 1:

1. Lag en ny fil som heter funksjoner.py og lagre den på skrivebordet. Kopier og lim inn programmet som tegner en likesidet trekant. Det finner dere her: https://tinyurl.com/trekant1
2. Endre på programmet slik at det benytter en funksjon for å tegne den likesidede trekanten.
3. Nå skal dere lage en ny funksjon i den samme fila. Denne skal tegne en annen type trekant enn likesidet. Dere bestemmer selv hvordan den skal være. Lag en plan før dere går i gang med kodingen.
   Her kan dere se ulike måter å skrive funksjoner på: https://tinyurl.com/funksjoner1
4. Endre på funksjonen slik at vi kan sende inn sidelengden til minst en av sidene eller vinklene i trekanten.
5. Utfordring: Vi kjenner to sider og vinkelen mellom sidene i en trekant. Trekanten er ikke rettvinklet. Lag en ny funksjon som tegner denne trekanten. Hint: Bruk trigonometri.

Innsending:

1. Legg inn en kommentar med navnene deres i programmet.
2. Sjåføren leverer inn programmet på teams.
3. Kartleseren leverer inn arkene.

Del 2:

1. Bytt program med en annen gruppe (send til hverandre på Teams).
2. Les programmet til den andre gruppa. Hva tror dere skjer når programmet kjøres? Diskuter i paret og skriv ned en hypotese i en kommentar i koden.
3. Kjør programmet.
4. Hva skjedde når dere kjørte koden? Hadde dere riktig i hypotesen deres?
5. Hvis det er noe dere ikke forstår i programmet, jobb for å forstå programmet med debuggingteknikkene på plakaten.
6. Har gruppa gjort noe annerledes enn dere? Fungerer programmene akkurat likt? Diskuter i paret.
7. Prøv å sende inn input som krasjer programmet til den andre gruppa. Hvorfor gir inputen feilmeldinger?
8. Prøv å sende inn input som ikke krasjer programmet, men som ikke tegner er trekant. Hvorfor blir det feil?

Økt 3:

1. Lag en ny fil som heter kunst.py og lagre den på skrivebordet. Kopier og lim inn funksjonen dere lagde som tegner en trekant. Fungerte ikke programmet deres helt, kopierer dere heller programmet som ligger ute på https://tinyurl.com/trekant2 .
2. Endre på programmet slik at det tegner trekanter forskjellige steder i tegneområdet. Lag en plan før dere går i gang med kodingen.
3. Endre på programmet slik at det tegner 10 trekanter forskjellige steder i tegneområdet.
4. Endre på programmet slik at det tegner trekanter helt til det samlede arealet av trekantene overstiger 10 000 kvadratpiksler.
5. La brukeren bestemme maksarealet.
6. Bruk farger for å gi litt liv til kunstverket!
7. Utfordring: Gå tilbake til funksjonen som tegner trekanter. Finnes det en annen måte å tegne trekanter på? Lag en ny trekant-funksjon og bruk den i kunstprogrammet.

Innsending:

1. Gi et nytt navn til fila slik at navnet passer til det som programmet gjør.
2. Legg inn en kommentar med navnene deres i programmet.
3. Sjåføren leverer inn programmet på teams.

K Live Coding in Session 2

```
import turtle

def areal(hoyde, bredde):
    areal = 0.5*bredde*hoyde
    return areal

def trekant(sidelengde):
    for i in range(3):
        turtle.forward(sidelengde)
        turtle.left(180 - 60)
trekant(200)
trekant(100)
A = areal(100,100)
print(A)
# Hindrer koden fra aa krasje
turtle.done()
turtle.bye()

```

Figure K.1: *Devolution. Note.* Result from the live coding session in the devolution phase in session 2. The program illustrates the implementation of subroutines with and without a return value in Python.

```
import turtle
import math

def rettvinklet_trekant(sidelengde):
    turtle.forward(sidelengde)
    turtle.left(90)
    turtle.forward(0.8*sidelengde)

    hyp = math.sqrt(sidelengde**2 + (0.8*sidelengde)**2)

    sin_alpha = sidelengde/hyp

    alpha = math.degrees(math.asin(sin_alpha))

    turtle.left(180 - alpha)
    turtle.forward(hyp)

rettvinklet_trekant(200)
rettvinklet_trekant(20)
turtle.done()
turtle.bye()

```

Figure K.2: *Institutionalization. Note.* Result from the live coding session in the institutionalization phase in session 2. The program illustrates the implementation of a subroutine drawing a triangle. The program also illustrates calls to the subroutine with different arguments.

L Transcription Codes

Underlined text	Text recited from the assignment the students are working on.
…	Break with a duration of maximum 3 seconds.
[…]	Skipping parts of the transcription.
- (hyphen)	Interruption.
(Text in round brackets)	Explanation of non-verbal action from the video or screen recording.
[Text in square brackets]	My own replacement of context dependent words such as “it” and “that”.

M Specialization Project Report

NTNU

Norwegian University of Science and Technology

Department of Computer Science

TDT4501 - Computer Science, Specialization Project

Integrating programming into *Matematikk 1T*

Candidate:
Anne Margrethe Vestgøte Bosch

Supervisor:
Majid Rouhani

December, 2021

Table of Contents

1 Introduction

1.1 Relevance

This study takes place within the context of the subject renewal in Norway, further referred to as LK20. LK20 is an update of the Norwegian National curriculum and changes the content of the subjects in schools (Meld. St. 28 (2015–2016)). In the autumn of 2020, the first changes were applied for the common core subjects in upper secondary school. The change relevant to this study is that programming has become part of the mathematics subject, including *Matematikk 1T* (Utdanningsdirektoratet [Udir], 2020). New competence aims are in place, but how the implementation will play out can first be seen through the operationalization of the curriculum in the classroom.

The introduction of programming in compulsory education is happening all over Europe. Programming is either integrated into existing school subjects, as a component in dedicated ICT subjects, or as one of several technologies implemented in all school subjects (Sevik, 2016). Programming is seen as an essential means to reach a larger goal - namely, to equip pupils with skills and competencies needed in the 21st century. There are different definitions of these competencies and skills (Sevik, 2016). The report *The School of the Future* (NOU 2015:8) is one of the catalysts for the subject renewal in Norway (Meld. St. 28 (2015–2016)). In the report, competencies for the 21st century are divided into four areas: competence in learning, competence in exploring and creating, competence in communicating, interacting and participating, and subject-specific competence (NOU 2015:8). The four competence areas equip students for the challenges of tomorrow's society and a labor market rapidly changing (NOU 2015:8). Programming has the potential for strengthening all four competence areas (Sevik, 2016). In particular, the potential for deeper understanding has been highlighted as a basis for including programming in the mathematics subject (Norstein & Haara, 2018). On the other hand, it has been argued that programming will amplify the problem of time pressure in mathematics courses and thus prevent in-depth learning (Sanne et al., 2016).

As the implementation is currently happening, there is insufficient research on the intersection between mathematics and programming in compulsory education on the upper secondary level. It has been hard to find related research in conjunction with this study. literature review on the use of programming in mathematics education from 2018 states that:

> As this literature review showed, sparse research exists on the educational potential of programming in mathematics education. [...] [W]e call for more research and research-based arguments in the policy for including programming in a mathematics education. (Forsström & Kaufmann, 2018, p. 28)

As clearly stated from the literature review, more research is needed in the field. This exploratory study aims to map out how in-service teachers currently in programming training are integrating programming into their instruction in *Matematikk 1T* in conjunction with the renewal of the curriculum.

The study contributes to the body of knowledge on approaches to implementation that are realistic and close to practice. This specialization project will provide a better understanding of how in-service teachers envision programming in their own mathematics classrooms. The analyses contribute to my subsequent master thesis, where I will create a lecture series design, test it in a classroom setting, and report on the results.

1.2 Problem statement and research questions

How do in-service teachers enrolled in programming training envisioning integrating programming into *Matematikk 1T*?

- RQ1: Which mathematics-related learning objectives do the teachers address?

- RQ 2: Which programming-related learning objectives do the teachers address?
- RQ3: Which programming-related student prerequisites are the lecture designs based on?
- RQ4: Which learning activities have teachers scheduled to obtain the learning objectives?

1.3 Structure

In chapter 2, the term *programming* will be defined and elaborated on, as it is a key concept in this study. The study combines a Constant Comparative Method for analysis (CCM) with a quasi-statistical approach. In chapter 3, the data material and methods for analysis applied in this study will be described in detail. Chapter 4, report on the findings from the analyses. The key finding is a model of three different approaches for implementing programming into *Matematikk 1T*. Chapter 5 revisits the research questions and discuss the findings. The following question is addressed through the discussion: *Are the programming-related learning objectives realistic?* Lastly, this paper suggests further research in the field intersecting mathematics and programming, and sets the stage for my master thesis.

2 Theory

As described in the introduction, this study is exploratory and does not build on a conceptual framework. Therefore, this section will reference literature to give definitions for the key concept used in this study: programming.

2.1 Three interrelated concepts: Programming, coding, and computational thinking

Programming is a critical concept in this study. What is programming? What is the difference between coding and programming? How does computational thinking relate to programming? There are numerous definitions of programming in the literature, and they vary in how broadly they define the term (Norstein & Haara, 2018). Through this study, it has also become evident that a proportionate amount of research does not explicitly define their notion of programming but rather lean on the tacit understanding of the reader, for instance, in a recent literature review regarding introductory programming (Luxton-Reilly et al., 2018).

This study will use a definition of programming close to the definition from a note created by the Norwegian Center for ICT in education (Sevik, 2016). Programming is a problem-solving process that results in computer programs written in a language (syntax) that can be understood computer. Part of the process is to understand and formulate the problem in a manner suitable to the context. The process also includes troubleshooting and iterative improvement of a program and evaluating solutions (Sevik, 2016, p. 9). A definition from a Norwegian organization is considered suitable as the study is situated in a Norwegian context.

A distinction between programming and coding is worth noting, as the terms coding and programming are frequently used interchangeably in the Norwegian discourse (Sevik, 2016). Coding can be understood as writing program code understandable to a computer. When the term is used in a Norwegian school context, coding has connotations of being less complex than programming and is limited to writing less advanced computer programs. Coding is an activity in the subset of programming, whereas programming also involves the problem-solving process of computational thinking (Norstein & Haara, 2018).

Programming and computational thinking are often discussed in conjunction with each other. One relevant example is the curriculum for *Matematikk 1T* (Utdanningsdirektoratet, 2020, p. 5). Computational thinking is creative and systematic problem-solving. It originates from Jeannette Wing. She defines computational thinking in a way that is close to this study's definition of programming:

> **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)

Computational thinking is understood in the renewed national curriculum (LK20) as a set of activities and key concepts in problem-solving. Programming is not mentioned in Udir's term elaboration, so computational thinking does not require writing program code (Utdanningsdirektoratet, 2019). Nevertheless, it is worth emphasizing that programming is often highlighted as an activity that fosters computational thinking (Norstein & Haara, 2018; Sevik, 2016).

As this elaboration conveys, the terms programming, coding, and computational thinking are complex, entangled, and used interchangeably in the Norwegian context. Further in this paper, I will consistently refer to the term programming and imply the whole process from problem identification to evaluation of solutions as described earlier. Nevertheless, the complexity is relevant to keep in mind as it complicates the scope and foundation of this study.

3 Methodology

3.1 Research design

The study I have conducted belongs to the qualitative research paradigm due to the research purposes and methods. The study does not intend to quantify general tendencies but instead tries to understand a contextualized situation through a small set of lecture designs. Equally important, the design is flexible and thus has evolved throughout the semester (Robson & McCartan, 2016). Fundamental to the study context, programming is in the process of implementation in the Norwegian national curriculum, as described in the introduction. Programming was introduced in *Matematikk* in 2020, and the lecture designs analyzed were created during the spring of 2021. Thus, the study is small-scale with case study traits (Robson & McCartan, 2016). The phenomenon to be researched in this case study is "implementation of programming in mathematics 1T".

The research design is influenced by the fact that this study is a preliminary project for my master's thesis. In my master thesis, I will investigate the same phenomenon further by designing and implementing a lecture series. The findings in this preliminary study will inform the future lecture series design.

3.2 Data material

The data is from a database of submitted assignments from in-service teachers currently in programming training. Further information about the population is described in section 3.2.1. The database contained 196 submissions from a programming course. Every submission contained a preliminary project, a lecture plan, a reflection note about the theme "programming for all," and a project report containing descriptions of the creating process and hours spent on the assignment. A majority of submissions included additional attachments such as code snippets. All data was anonymized when it was accessed in August 2021.

In this study, only the preliminary projects (PPs) are analyzed due to the scope of this study. The PPs are planning documents outlining lecture designs without including every detail. The PPs are written documents based on a template. The template contains the sections: The main topic of the lecture, target subject(s), competence aims, learning objectives, prerequisites, short overall project description, lecture activities, lecture implementation plan, time frame, and a list of tools. The template is included in Appendix A.

21 PPs for lecture designs created were analyzed. 16 of the 21 projects were submitted individually, and five were submitted by two teachers collaborating. Hence, 26 teachers contributed in total. Section 3.3 describes the filtering process of the data material from 196 to 21 PPs.

3.2.1 Population

The PPs analyzed are created by in-service teachers enrolled in the teacher training course IT6204 - Applied Programming for Teachers. The course is web-based and provided at the Centre for Continuing Education and Professional Development in the Norwegian University of Science and Technology (NTNU) in Trondheim (NTNU, n.d.). The teachers have already completed the course IT6203 - Introductory Programming for Teachers, giving them 15ECTS of programming training from the two courses. As the course is web-based, participants can be located in all parts of Norway. Teachers teaching all school subjects spanning primary through upper secondary level partake in the course (Rouhani et al., 2021; Rouhani et al., 2019). However, only mathematics teachers teaching the specific subject *Matematikk 1T* are investigated in this study. As programming is a new addition to the standard curriculum for mathematics, teacher training is needed to equip teachers with the necessary knowledge and skills (Sanne et al., 2016, p. 77). Consequently, the population of this study is likely more proficient in programming and programming didactics than the average mathematics teacher in Norway. Hence, the population could be regarded as a limitation, as addressed in section 5.5.

3.3 Data organization and initial filtering

The first step of analyzing qualitative data is organizing the data material to keep it manageable. Organizing is part of the analysis, as choices on how to structure the data is also a process that shapes the end product (Robson & McCartan, 2016, p. 466). When I was granted access to the data through my supervisor's database, the entries were sorted in folders by candidate numbers. During the first scan of the data material, the 196 database entries were systematically visited based on ascending candidate numbers. The preliminary projects were read through, and relevant data about each preliminary project were manually extracted to a spreadsheet. The columns of the spreadsheet are displayed in Table 1.

Before and during this scan, inclusion and exclusion criteria were developed to reduce the amount of data. The final inclusion and exclusion criteria from the first scan of the data material are listed below.

Inclusion criteria

- Mathematics included as a subject in the PP
- Lower secondary or upper secondary school level

Exclusion criteria

- Duplicate projects due to teacher collaboration
- Alternative curriculum
- Not concerning mathematics subject(s)
- Preliminary project missing from database entry
- Primary school level

After the initial filtering, the number of PPs was reduced from 196 to 72. In accordance with the inclusion and exclusion criteria, the 72 PPs addressed all mathematics school subjects on the lower and upper secondary levels. The amount of data was still too considerable for the scope of this specialization project. Therefore, the scope got reduced further to one mathematics subject for further investigations, namely *Matematikk 1T*. This subject is theoretical mathematics for the first year of upper secondary school. The subject is available to pupils who intend to proceed to higher education and is an alternative to practical mathematics, *Matmatikk 1P*. The selected data material for further analysis was now 24 PPs regarding *Matematikk 1T*.

The choice of mathematics subject to proceed with was grounded in personal experience. I have previously researched the implementation of *Matematikk 1T* before from a teacher perspective in a pilot project. Lower secondary mathematics could also have been selected, as there were 28 entries in the database on that level. Other mathematics subjects on the upper secondary level were sparsely represented in the database.

Spreadsheet column	Explanation
Candidate number	Candidate number on the database
Collaborators	Candidate number(s) of collaborators, if any
Subject	Main subject from the Norwegian school system pertinent
Subject code	Main subject code. Retrieved from UDIR's website.
Additional subject(s)	Additional subjects pertinent, if any. Cross-curricular teaching.
Additional subject code(s)	Retrieved from UDIR's website
Grade level	Either primary, 8-10, vg1, vg2, vg3, or a combination of these.
Technology	Specific technologies pertinent, if any. For example Micro:bit or Python
Approach	Programming approach, either block based, text based, or a combination of the two
Formal deficiencies	Excluding factors such as author name (non-anonymized), preliminary project missing in the delivery, wrong candidate number, empty delivery etc.
Relevant in round 1?	Column with only yes or no as possible answers based on inclusion and exclusion criteria

Table 1: Spreadsheet columns used in the data filtering process. The right column of the table gives a brief explanation for each of the spreadsheet columns' contents.

3.4 Analysis

After the initial data selection, the data has been approached for analysis in two different ways. The primary method for data analysis is the *method of constant comparison* (CCM) inspired from *grounded theory*. Through this approach, themes are identified in the data material inductively (Robson & McCartan, 2016, pp. 481-484). The method of constant comparison will be further described in section 3.4.2. An additional analysis was conducted to support the CCM analysis. This additional approach is quasi-statistical, where aspects of the qualitative data have been quantified through descriptive statistics (Robson & McCartan, 2016, p. 461). This analysis will be further described in section 3.4.1.

3.4.1 Descriptive statistics

Different frequencies have been recorded regarding duration, competence aims, tools, languages, and environments (TLEs), mathematical topics, and collaboration from the data material. The purpose of the quantification and graphical presentation of aspects of the data is to understand similarities and differences within the entire data set.

Both measures of central tendencies and measures of variability are used to analyze the data. Mean, median and mode are the most common to measure central tendencies (Robson & McCartan, 2016, p. 419) and have been used to explore lecture duration. The results are described in section 4.1.1. Suitable diagrams and tables are included to display central tendencies and variance within the data, such as occurrence of competence aims.

As the data set is relatively small, I have manually plotted the pertinent data to an Excel spreadsheet instead of applying more sophisticated software. Built-in Excel formulas were used to calculate the measures of central tendencies and generate diagrams.

3.4.2 Constant comparative method

The constant comparative method of analyses (CCM) is applied in this study. CCM has its origins in *grounded theory*, a rigorous methodological approach for data-driven inductive qualitative research, but CCM is also suitable for other flexible research designs such as case studies (Postholm, 2005, p. 87). In CCM, the analysis process is divided into three phases: open coding, axial coding, and selective coding. The goals are respectively for the three phases to label similar meaningful data segments with codes, group the codes into categories, and abstract the categories into one or more core categories that encapsulate the essence of the findings from the data material (Postholm, 2005, pp. 87-91; Robson and McCartan, 2016, pp. 481-484). The starting point of CCM is open coding, but the process of coding, category identification and core-category identification is not linear. The process is a *constant comparison* between data segments, where new codes and categories can emerge, merge and be restructured throughout the whole analysis process (Robson & McCartan, 2016, pp. 481-483).

CCM was chosen due to the study's exploratory nature, where a suitable theoretical framework for analysis would not be appropriate to define in advance. A data-driven approach through CCM was considered more appropriate.

Open coding

This was the first step of the CCM data analysis. Data was divided into discrete segments called codes. The researcher decide the coding unit size (Robson & McCartan, 2016, p. 481). In this study, the code unit size varies from single words to paragraphs depending on the type of meaning carried in the segment. For example, the segment contianing the name "Micro:bit" is coded to "Sensor to register data", whereas a whole paragraph describing students' programming prerequisites is coded to "Builds on basic programming skills". Data segments can be coded into several codes (Robson & McCartan, 2016, p. 481), which is done in this this study.

The 24 PPs selected for analysis were added to an NVivo project and coding was carried out through NVivo's coding functionality. The PPs were sequentially coded based on candidate number. During this process, another three PPs were excluded from the data material. Two were excluded because they were cross-curricular and mainly concerned with natural science, whereas mathematical learning objectives were stated in the PPs as a positive side-effect. The third PP was excluded because it addressed competence aims from the outdated national curriculum. After the initial round of open coding, the 21 remaining PPs were revisited for another round of coding. The second round of coding was carried out because new codes were identified late in the coding process, and had to be taken into consideration for all PPs.

146 codes were identified through the open coding process. Table 2 displays some of the most frequently occurring codes, and examples of associated data segments. Not every identified code is relevant to the findings discussed in this specialization project, which is in line with the CCM process (Robson & McCartan, 2016, pp. 481-483). For transparency, the whole coding scheme hierarchy created in NVivo is displayed in Appendix B.

As described in Robson and McCartan (2016), when several researchers are coding the same data material and compare coding, it reduces subjectivity and bias. In this project, the coding is carried out by only one researcher, which leads to a potential bias. A means taken to create more transparency in the process was a meeting to discuss the coding with a supervisor after the initial code set was created. Codes and data segments associated with the codes were discussed during the meeting. I address this limitation in section 5.5.

Code name	Representative data segment
Duration	Session 1 (90 minutes): Pythagoras and basic programming. Session 2 (90 minutes): Basic trigonometry, i.e. relationship between sides and angles. Find [sides and angles].
Programming to enhance mathematics learning	Pupils are to be able to use numerical methods for investigation of the relationship between average growth rate and instantaneous growth rate.
Builds on basic programming	They should also have worked with some programming before, as the program in its full form is a relatively large and complex program.
Competence aim 1	[F]ormulate and solve problems through the use of algorithmic thinking, different problem solving strategies, digital tools and programming
Loops	[. . .]to create while- or for-loops that prevent the program from just going in circles [. . .]

Table 2: The table gives examples of codes frequently occuring in the data material and associated representative segments. The data excepts are translated from Norwegian to English.

Axial coding

In the axial coding process, codes were organized into categories and subcategories. This was done in NVivo, restructuring the set of codes into parent and child codes. Because of limitations in the NVivo tool, codes could only be categorized into *one* category. Axial coding was primarily done after the open coding, but the common themes such as the different approaches were emerging already during the initial coding. The categories identified are displayed in Table 3 together with associated subcategories and codes.

Selective coding

The last part of the CCM analysis process was selective coding, where a core category is identified from the most pertinent theme spanning the categories. The core category is an umbrella term to conceptualize the overall picture painted from the findings in the data material (Robson & McCartan, 2016, p. 483). From this study, the core category identified is "Approaches to implementing programming into *Matematikk 1T*". The core category will be thoroughly explained in section 4 as it is the main finding of this study. The process of identifying the core category was the most time consuming part of the CCM analysis. It was conducted through constant comparison between codes and categories in NVivo, in parallel with drawing models on paper.

3.5 Ethical considerations

The study, as other document analysis studies, is unobtrusive research (Robson & McCartan, 2016, pp. 346-348). Due to the unobtrusive nature of this study, the researcher has not affected the population studies. Another ethical aspect to highlight is that the participants are not identifiable from the report, as the analyzed data was fully anonymized before access was granted. Furthermore, the nature of the findings in the study is not harmful to anyone. Thus, the study is in line with the *Guidelines for Research Ethics in the Social Sciences, Humanities, Law and Theology* provided by The National Committee for Research Ethics in the Social Sciences and the Humanities (NESH) (2019).

Category name	Examples of associated (subcategories) and codes
Approach	Programming to enhance mathematics learning Builds on basic programming skills Learn basic programming Builds on unspecified programming skills Exams Builds on little or no programming skills ...
Learning objectives	Content knowledge Mathematical topics Trigonometry ... Programming topics Loops Conditionals Output or print ...
Regulations from the national curriculum	Competence aims 1T Aim 1 ... Aim 14
Tools	Personal computer Hand writing GeoGebra Theory Book Google Colab Spyder YouTube ...

Table 3: Categories identified in the data material through the axial coding process. The right column in the table provides examples of associated subcategories and codes for each of the categories.

4 Results

Through the data analysis, I have compared content from the whole data set. These findings describe duration of the lectures, pertinent competence aims, and tools, languages and environments (TLEs).

I addition to these descriptive analyses on the whole data set, I have developed several models describing the different approaches to teaching programming identified in the data material. The first model describes the contents of an approach, that is the actual lecture (series) design, and factors that seems to be influencing the choice of approach. Only factors available or partially available from the data set are included. See 4.

The second model describes the three approaches identified, and how they seem to relate to each other. See 7.

4.1 Findings from descriptive statistics

4.1.1 Duration

Of the 21 PPs, 18 specify the number of hours intended to spend on programming in *Matematikk 1T*. Mode and median were both $6h$, mean was $6,97h \approx 7h$. The maximum value was $12,75h$ and

the minimum was $3h$, which gives a range of $9,75h$. Figure 1 shows a histogram of the frequency within the 18 PPs that reported intended hours. The three PPs that did not specify the number of hours described duration of three periods, three lectures and five lectures respectively, where the duration of a period or lecture is undefined.

The total number of allocated hours within a school year in *Matematikk 1T* is $140h$ (Utdanningsdirektoratet, 2020). Comparing the mean duration of $7h$ in the data material to the total number of hours, the mean duration of programming in *Matematikk 1T* is 5 % of the total number of hours in the subject.

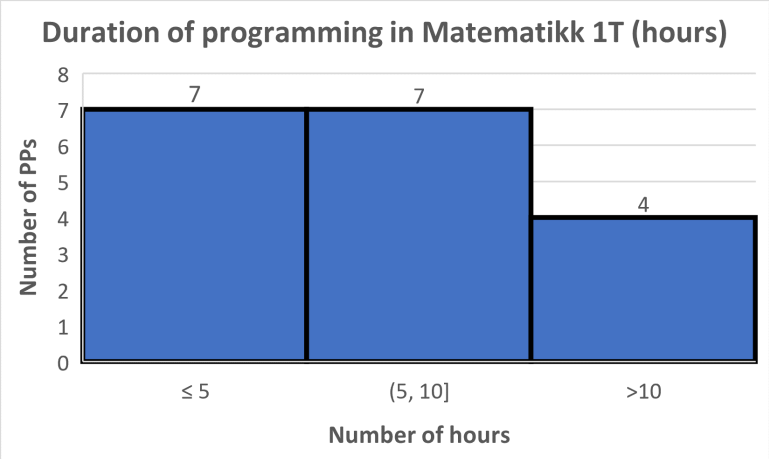


Figure 1: Histogram showing the number of hours intended for programming in the preliminary projects (PPs). Bin size = 5 hours. The assumption is made that no lectures can have a duration less than 1 hour.

4.1.2 Competence aims

All of the preliminary projects describe which competence aims from LK20 the teacher(s) consider relevant for their lectures. I have enumerated the competence aims according to the order they are listed in the *Matematikk 1T* curriculum (Utdanningsdirektoratet, 2020). In total there are 14 competence aims. The enumerated list is displayed here:

The pupil is expected to be able to

1. formulate and solve problems through the use of algorithmic thinking, different problem solving strategies, digital tools and programming
2. read and understand mathematical proofs and explore and develop proofs in relevant mathematical topics
3. identify variable amounts or magnitudes in different situations, create formulas and explore these using digital tools
4. explore strategies for solving equations, systems of equations and inequalities, and argue for one's thought processes
5. explain the difference between an identity, an equation, an algebraic expression and a function
6. explore connections between quadratic equations, quadratic inequalities, quadratic functions and quadratic formulas, and use these connections in problem solving

7. model situations in relation to different topics, discuss, present and explain the results and then argue the validity of the models
8. read, extract and assess mathematics in relevant texts on various topics and then present relevant calculations and analyses of the results
9. explore and describe the properties of polynomial functions, rational functions, exponential functions and power functions
10. use average and instantaneous growth rates in concrete examples and then account for the derivative
11. explain polynomial division and use it to rewrite algebraic expressions (factoring), discuss functions and solve equations and inequalities
12. explain the definitions of sine, cosine and tangent and use trigonometry to calculate the length, angles and area of random triangles
13. explain the area, sine and cosine rules
14. use trigonometry to analyse and solve complex theoretical and practical problems through length, angles and area

(UDIR, 2020, p. 5)

Some competence aims occur more frequently in the data material than others. As can be seen in Figure 2, competence aim 1 is included in all of the preliminary projects analyzed. This competence aim is the only one of the 14 to explicitly include the term "programming". Competence aim 3 is the second most pertinent, and occurs in 13 of the preliminary projects. Aim 1 and 3 are the only two competence aims of the 14 to include the term "digital tools". Both aim 7 and 14 are referenced in 9 preliminary projects, and are the third most pertinent competence aims in the data set.

From Figure 2, it is also apparent that most of the competence aims from the *Matematikk 1T* are referenced at least once within the data set. From the 21 lecture plans, only aim 5 and 11 are not referenced at all. This finding implies that most of the curriculum in *Matematikk 1T* can be connected to programming lectures.

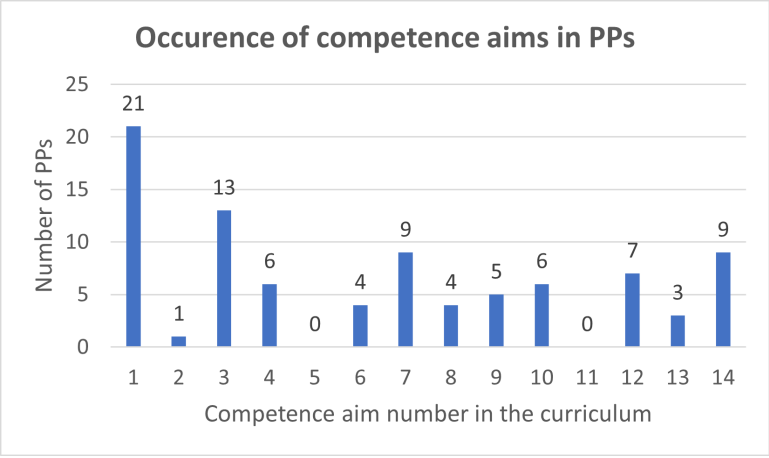


Figure 2: Bar chart showing the occurrence of competence aims from the curriculum in the preliminary projects (PPs). Competence aims are enumerated as described above.

Another finding is regarding the number of competence aims referenced in each preliminary project in the data set. Figure 3, the most frequent number of referenced competence aims are four. No less than two, and no more than seven competence aim s are included as relevant in a single preliminary project. This finding also supports the statement that teachers span a variety of topics within their planned lectures.

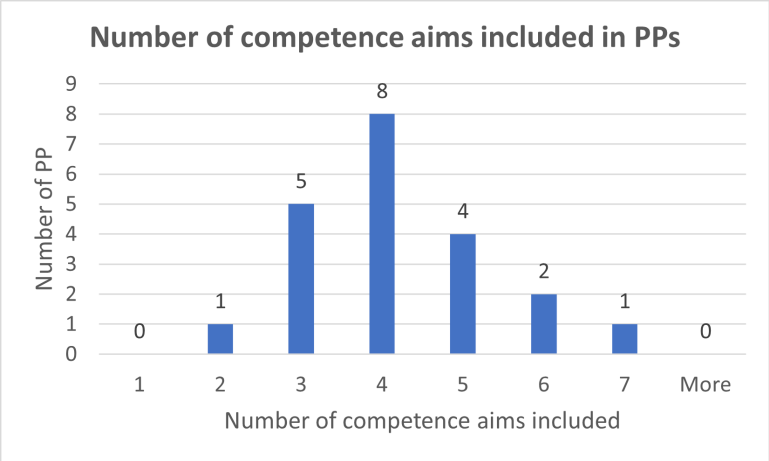


Figure 3: Bar chart showing the number of competence aims included in each of the preliminary projects (PPs). The assumption is made that lectures can not reference 0 competence aims.

4.1.3 Mathematical topics

The PPs are for lectures in a mathematics course. Hence, the learning of mathematics is in focus for most of the PPs. Some of the projects focus more on learning introductory programming than mathematics. I will elaborate on this in section 4.2.1.

Table 4 displays what the main mathematical topics in each of the 21 lecture plans are. Introductory programming is included, due to the fact that programming is mentioned explicitly in one of the competence aims (Utdanningsdirektoratet, 2020). As can be seen from table 4, trigonometry is the most frequent main mathematical topic from the data set with eight PPs.

Not every PP addresses only one mathematical topic. As shown in table 5, the sum of mathematical topics in focus exceed 21. The most frequent addressed mathematical topic is trigonometry, with nine PPs.

Number of PPs	**Main mathematical topic**
8	Trigonometry
6	Problem solving and introductory programming
5	Differentiation
1	Number theory
1	Analysis
Sum: 21	

Table 4: Table showing the main mathematical topic addressed in each of the preliminary projects (PPs).

Number of PPs	Mathematical topics
9	Trigonometry
7	Introductory programming and problem solving
6	Differentiation
3	Figurate numbers
2	Analysis
1	Number theory
Sum: 28	

Table 5: Table showing the number of preliminary projects (PPs) that address different mathematical topics. One PP can address several mathematical topics, hence the sum exceeds 21 PPs.

4.1.4 Tools, languages and environments

All of the preliminary projects make use of programming on personal computers, one computer per student. In this section I will describe the tools, languages and environments (TLEs) present in the preliminary projects, both digital and non-digital. The findings from this section will not be discussed in section 5, but will inform the design decisions in my subsequent master thesis.

Python programming

Text based programming in Python is included in all preliminary projects. Python is accompanied by integrated development environments such as Anaconda Spyder, Jupyter Notebook, PyCharm, Google Colab or Trinket. The selected programming environment for Python programming is not described at all in eight of the PPs. Only one PP describes the use of several different environments for Python programming, that is both Anaconda Spyder and Jupyter Notebook.

Supplementing programming TLEs

Other TLEs than Python are supplementing the use of Python. Three PPs describe logging data with sensors, where one of them explicitly states the use of BBC Micro:bit to this end, another make use of Pasco Smart Cart, and the third does not specify the selected tool(s). Two PPs are concerned with block based robot programming. One of them make use of Fable robots, the other Sphero RVR robots.

Non-programming digital tools

In addition to different programming modalities, several of the PPs make use of other digital tools for supporting the learning activities. GeoGebra computer algebra system (CAS) is used in two PPs. GeoGebra Graphing Calculator is used in six PPs. Microsoft Excel is used in one PP, YouTube videos are used in one PP, and traditional calculators are used in one PP.

Non-digital tools

Not only digital tools are used in the PPs. Handwriting is made use of in 13 PPs. The theory book is references as a tool in three PPs, and other forms of written material such as worksheets are to be handed out to students in three PPs. Counting chips and measuring tape are made use of in one PP each.

4.2 Three identified approaches

Through the constant comparative analysis of the data material, I have developed a model of different approaches to integrating programming in mathematics 1T. Hence, the core category identified through the data material is "Approaches to integrating programming into *Matematikk 1T*". The name of the core category refers to the overall approach the teachers apply for their lectures. First, I will describe what are the components of an approach in general. Afterwards, I will describe the characteristics of the three specific approaches identified and how they relate to each other.

4.2.1 Components of an approach

As displayed through the model, the lecture design in the center of the model consists of subcategories that constituents the lecture design. Several contextual factors are influencing the lecture design. In the model, these are described in the bubbles surrounding the lecture design category. The model is not complete. Not all influencing factors are included. Only those who are identifiable (solid border) or partly identifiable (dotted border) through the data material are included.

I will describe findings from the data material regarding lecture design components, and relate the findings to the different approaches identified in the next section, 4.2.

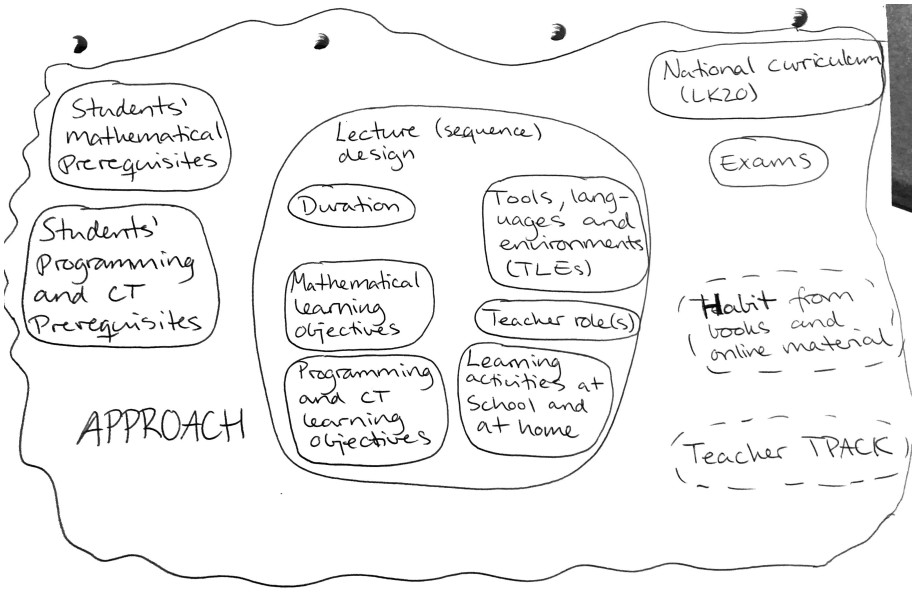


Figure 4: Model showing the components of an approach to integrating programming into *Matematikk 1T*. The lecture (sequence) design and its components is the area in the middle of the model. Influencing factors for the design are surrounding the lecture (sequence) design area. Dotted lines denotes influencing factors only partially identifiable through the data material in this study.

4.2.2 Approach 1 – Learning (introductory) programming

One approach is to teach the first steps of introductory programming through mathematical topics. Two of the preliminary projects from the data material take this approach. They build the lecture on the assumption that their students have no or very little prior experience with programming. One teacher elaborates on the selected approach in their preliminary project description:

> We expect that the class starting in the autumn of 2021 also will have little knowledge in programming, as they haven't followed the new curriculum. We also assume that it will take several years before the students who come to upper secondary school have good enough knowledge in programming so that we can directly make use of it for learning mathematics. There will probably be large variations in the type of programming they have experience with from lower secondary school. (Quote from Preliminary project number 10100)

The other 19 preliminary projects build on the assumption that students have some experience

with programming, hence they build on approach 1 either from previous lectures in the same school year, or from lower secondary school.

Learning objectives within this approach I have conducted two queries in NVivo to answer this question. First, I ran a matrix query between the code "Builds on little or no programming skills" and all codes within the category "Learning objectives/Context knowledge/Programming topics" with a NEAR criteria on the common file leven. The second query was "Builds on basic programming skills" AND "Learning objectives/Context knowledge/Programming topics", where the results contained only overlaps. Through the queries I wanted to identify the most frequent learning topics to be learned through approach 1. What I can tell from the data material is limited. Of the 19 preliminary projects that build on approach 1, only 10 specify the concepts they expect their students to be familiar with. Furthermore, the two preliminary projects taking approach 1 gave more thorough descriptions of the expected learning outcomes than the 10 that only builds on the approach.

The programming concepts that are expected for students to learn through approach 1 identified through the queries are described here. The most frequent occurring topics were definite and indefinite loops, conditionals and variables. The topics occurred in respectively nine, nine and seven of the 12 preliminary projects queried. Other topics that quite frequently occurred were defining and calling functions, textual output through print, primitive data types, arithmetic operators, and import and use of third party libraries. Topics that were mentioned in only one or two preliminary projects were recursion, what is an algorithm, debugging, composite data types such as lists, user interaction through the input function, and visual output through graphs.

Codes from the set of programming topics codes that did not appear in the queries were flowcharts, iterative improvement of code, pseudo code, reuse of code, robot programming, testing, and write understandable code.

Duration One of the lecture sequences described had a duration of 12 hours planned. The other one does not specify the number of hours, but is integrated into the lectures throughout one semester.

4.2.3 Approach 2 – Apply programming to learn mathematics

Six of the preliminary projects (PPs) from the data material had an approach where they applied programming in the context of mathematics. The identifier for these projects was the lack of the code "Learn basic programming"'. In these projects, there is no or very little emphasis on learning new programming concepts or syntax during the lectures. Applying existing programming knowledge to enhance mathematics is the explicit main objective. The two examples below illustrate the focus on mathematical learning objectives through the programming medium:

> Differentiation: The main goal of the lecture series is to work with average and instantaneous growth rates and transfer [the knowledge] to numerical differentiation. In the code part of the lectures, we work mainly with numerical differentiation, preceding this we will work with understanding average and instantaneous rate of change in more traditional forms with paper and pencil and by illustrating and talking about connections in GeoGebra. (Emphasis added. From PP 10115)

> Trigonometry: The students will program a "triangle solver" in Python. Through the work, they will have to apply knowledge in trigonometry from Mathematics 1T to solve the problem. The aim is to gain a thorough understanding of concepts from trigonometry and to be able to apply trigonometric formulas. (Emphasis added. From PP 10040)

Mathematical topics All of the 6 PPs addressed only one mathematical topic each. Three of the preliminary projects were concerned with trigonometry, two with differentiation, and one with number theory.

Builds on basic programming Of the 6 lecture plans, only two of them identified exactly what programming prerequisites they expected from their students. Both these were concerned with trigonometry. The programming concepts they expected their students to know were

1. Conditionals, definite and indefinite loops and textual output through print
2. Conditionals, definite and indefinite loops, variables, and defining and calling functions

4.2.4 Approach 3 – Dual focus on learning programming and mathematics

The most common approach identified through the data analysis was a dual focus on learning new programming concepts, as well as applying previous and new programming concepts in the learning of new mathematics. In other words, the third approach is uniting the two previous approaches. From the data material, the teachers either alternated between the approaches, or merged them into a single teaching unit. I have therefore divided approach 3 into two variants.

Alternating variant The first variant is alternating between activities focusing on introductory programming and activities focusing on applications of programming into mathematics. The following quote exemplifies the alternating variant:

> First, the pupils repeat the most necessary programming concepts, before using them to create programs they can use to draw function graphs, find average and instantaneous growth rates, and draw the graph of the function's derivative. The pupils will also create programs that can be used to solve equations. (Colors added. Red for introductory programming focus. Blue for application focus. From PP 10016)

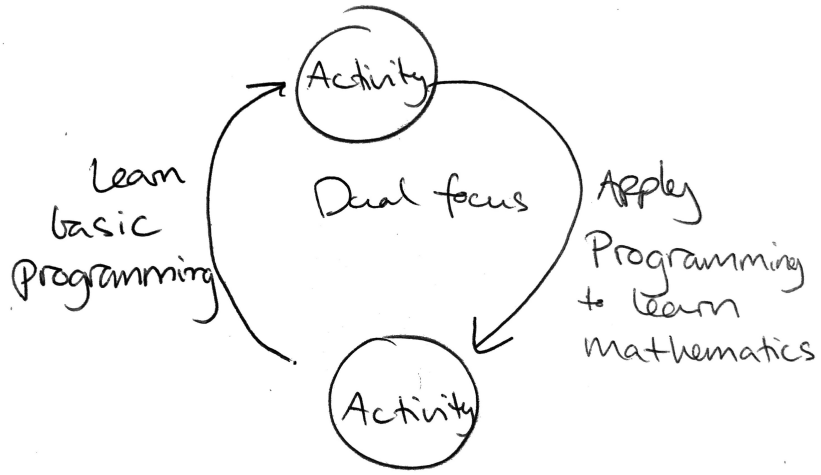


Figure 5: Model illustrating the alternating variant of the dual focus approach. Activities alternate between targeting on learning objectives focusing on learning basic programming and applying programming to learn mathematics.

Integrated variant The other variant is integrating both new programming concepts and applications of the programming concepts into the same activity. The following quote from the data material exemplifies the variant:

> What is new for the pupils in this lecture is to define and plot graphs and to use the programming in connection with growth rate and differentiation. [...] The pupils

> create a program that draws function graphs in Python. Furthermore, the program should be extended to analyzing the function by finding the average growth rate in an interval, the derivative of the function by numerical derivation, and the expression for the tangent line to the function for a given x-value. The graph of the derivative must also be plotted so that the students can see connections between the two graphs. [...] (Colours added. Red for introductory programming focus. Blue for application focus. Purple for the activities that integrate both rationales. From PP 10034)

In this preliminary project, the teachers also elaborate on their rationale for choosing the approach:

> We think it is important that there is not too much new introduced to the pupils at once, especially when they only have short experience with programming. (From PP 10034)

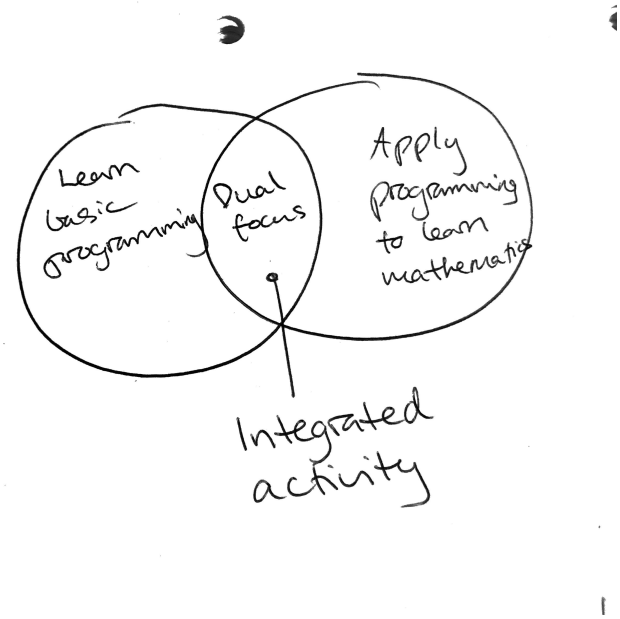


Figure 6: Model illustrating the integrated variant of the dual focus approach. Activities target learning objectives focusing on both learning basic programming and applying programming to learn mathematics.

4.2.5 How the three approaches are related

Figure 7 gives an illustration of how the three identified approaches are related in regards to learning objectives within programming and mathematics.

	Are the programming concepts new?		
		No	Yes
Are the mathematical concepts new?	No		Approach 1
	Yes	Approach 2	Approach 3

Figure 7: Model illustrating how the three approaches to integrating programming into *Matematikk 1T* are related. Learning objectives based on programming and learning objectives based on mathematics are the basis for comparison of the approaches.

5 Discussion

As a result of the subject renewal (LK 20), the Norwegian mathematics curriculum has changed to include programming. This study aims to give a better understanding of the current implementation of programming into *Matematikk 1T* on the upper secondary level. This study gives insight into how in-service teachers enrolled in programming training envisions the implementation in their own classroom. The investigation is conducted through analysis of the teachers' preliminary projects (PPs) for lecture designs. The research questions I have explored in this study are:

- RQ1: Which mathematics-related learning objectives do the teachers address?
- RQ 2: Which programming-related learning objectives do the teachers address?
- RQ3: Which programming-related student prerequisites are the lecture designs based on?
- RQ4: Which learning activities have teachers scheduled to obtain the learning objectives?

5.1 RQ4 – Learning activites classified through three approaches

The model created through this study classifies the scheduled learning activities into three approaches. Either the teaching activities address programming-related learning objectives, apply programming to enhance mathematics, or a combination of both. Prerequisites have an influence on the choice of approach. This model is the main finding from this study. Through research questions RQ1, RQ2 and RQ3 the learning objectives and prerequisites included into the three approaches will be discussed further.

5.2 RQ1 – Mathematics-related learning objectives

Figure 2 showed that within the 21 PPs, 12 of the 14 competence aims from the National curriculum were represented. The most frequently occurring mathematical topic was Trigonometry. Differential calculus and introductory programming was also frequently present as mathematics-related learning objectives.

A pertinent question to address in relation to the mathematical topics in focus is whether introductory programming could be regarded as learning mathematics or not. I have made the assumption that programming is included into the scope of "mathematics" on the basis that the curriculum in *Matematikk 1T* includes programming in competence aim 1, as shown in section 4.1.2. This topic needs to be discussed further as influences the role of programming in mathematics education. One example of the influence is the three approaches described in this paper, where both approach 1 and 3 include classroom activities where new programming concepts is taught. Are these activities within the area of responsibility for mathematics?

5.3 RQ3 – Programming-related student prerequisites

As described in section 4.2.1, the approach chosen by a teacher is influenced by pupils' programming prerequisites. One variation seen in the data material is that the teachers build their lectures on the assumption that pupils do not know any programming. The argumentation is closely tied to the fact that programming in *Matematikk 1T* is still in the implementation process. Hence, this approach is relevant as of now, but as students get more programming prerequisites in the future from earlier school stages, it will become less relevant.

The other possibility is that teachers build their lectures on pupil programming prerequisites. They either refer to a set of basic programming skills without stating the contents of such a set, or they explicitly spell out which prerequisites are required or expected.

As seen through the three approaches, student prerequisites influence the approach to integrating programming into mathematics. Programming-related prerequisites are closely tied with programming-related learning objectives. This will be further discussed in 5.4.

5.4 RQ2 – Programming-related learning objectives

15 of the 21 PPs in the data material address programming-related learning objectives. These are the PPs categorized to Approach 1 and Approach 3. In Approach 2, the main goal is to apply programming in order to enhance mathematics learning. As already discussed, depending on student prerequisites, among other influencing factors, teachers choose one of the three approaches, and adjust the programming-related learning objectives accordingly. When students have obtained the required prerequisites, the mathematics-related learning objectives gets the main focus, as displayed through Approach 2. Consequently, the set of basic programming skills that makes up the basis for applying programming seems important to elaborate on.

I will discuss the specific programming-related learning objectives that make up the foundation and compare them to other studies. The list below displays the programming topics considered as a foundation in the data material. All the topics are never covered within a single lecture design, and the frequency of occurrence is decreasing further down the list.

- Definite loops
- Indefinite loops
- Conditionals
- Variables
- Defining and calling functions
- Textual output (print)
- Primitive data types
- Arithmetic operators
- Import and use of third party libraries
- Recursion
- What is an algorithm
- Debugging
- Composite data types such as lists
- User interaction (input function)
- Visual output (graph plotting).

This finding is relatively close in line with previous research on which components make up the core of basic programming. A thorough process from 2010 identified the ten most critical topics of an introductory programming course on the university level:

- Fundamentals: Variables, assignments, expressions
- Logical operators
- Testing and selecting alternatives (e.g., `if` statements)
- Definite loops (`for` loops)

- Indefinite loops (`while` loops)
- Arrays
- Functions and methods with parameters
- Functions and methods with return values
- Recursion
- Object-oriented basics (e.g. reading a class definition and calling methods on an object)

(Guzdial, 2015, pp. 25-26)

When comparing the findings from this study to the 2010 overview, I have considered Arrays to be analogous to Lists, and Logical operators and selecting alternatives to be included into Conditionals, as if statements are built on logical operators. When comparing the 2010 overview to the findings, the most pertinent differences are object oriented programming and testing. These two concepts are not considered within the scope of basic programming in this study. When comparing the other way, it gets more interesting. There are several topics recurring in the data material that are not on the 2010 top ten list. Some of these seem fundamental: what is an algorithm, textual output (print), arithmetic operators and user interaction (input). Other topics seem less fundamental: Import and use of third party libraries, debugging and visual output (graph plotting). The differences are probably influenced by the fact that the 2010 overview is concerned with introductory computer science classes (CS1), whereas the data material in this study is from mathematics classes. Nevertheless, the content is strikingly similar.

The time dedicated to learning the building blocks of programming is another aspect to take into consideration. As described in section 4.1.1, the average time dedicated to programming in the data material was $7h$ of the total $140h$ in a year that make up *Matematikk 1T*. It is a mathematics subject, and the curriculum is made up of 14 competence aims where only two address digital tools, and one addresses programming as seen in section 4.1.2. The CS1 course described in Guzdial (2015, pp. 24-27) also consisted of one year of classes. The total number of hours is unknown. Nevertheless, the 10 programming concepts described above make up the *essential* learning objectives expected to be obtained by young adults throughout the course.

This points to the question whether the programming prerequisites expected from pupils in *Matematikk 1T* are realistic or not. As both Guzdial (2015) and Luxton-Reilly et al. (2018) repeatedly stresses, realistic expectations is crucial to make programming more available to everyone, as well as getting rid of the assumption that programming is inherently hard. The question sets the stage for my master thesis. Through this preliminary study, I have mapped out three approaches to teaching programming and their associated prerequisites and learning objectives. I have identified which mathematical topics the teachers envision connects most closely to programming in *Matematikk 1T*. The next step is designing series of lectures informed by this preliminary study, test out the lectures in the classroom and report on the findings. That will be one step closer to investigating which programming expectations we can have for students when they start *Matematikk 1T* and when they finish *Matematikk 1T*.

5.5 Limitations

A small sample of preliminary projects for lecture designs are analyzed in this study. Analyzing a larger number of PPs has the potential to reveal more details about the three approaches identified, such as the most pertinent mathematical topics in each approach. Data triangulation through interviews with the teachers who created the lecture designs would also give insight into the rationales behind the lecture designs. Due to anonymization, it was not possible to identify and contact any of the contributing teachers.

Furthermore, the population can be considered both a strength and a weakness of the study. The teachers who have created the lecture designs were all enrolled in programming training. This is a

strength because the population makes informed decisions in regard to programming, but it also raises questions whether the results would be the same if a random sample of teachers created the lecture designs. It is a possible direction for further research to do the same analysis with lecture design created by the more general population of mathematics teachers teaching *Matematikk 1T*.

Another limitation of the study worth pointing out is the fact that only one researcher has conducted the study. Especially the coding process would benefit from a more rigorous and transparent process of two coders collaborating and comparing their results (Robson & McCartan, 2016). To counter the researcher bias, I have arranged a meeting to discuss codes and coded segments with my supervisor. However, the results might still be subject to researcher bias.

Lastly, this is an interdisciplinary study between mathematics and programming. Whereas the term programming has been rigorously defined in the theory section 2.1, the term *mathematics* and topics within mathematics such as trigonometry and differentiation has not been defined or discussed in the same manner. The understanding of these concepts lean on the implicit understanding of the reader and the researcher. A more rigorous investigation of the terms did not fit into the scope of this study. To strengthen the reliability of the results, such investigation would be appropriate.

5.6 Further research

Several questions have emerged from the investigations. One of the contributions of this paper has been to identify which programming skills and knowledge is expected for pupils to obtain through the course *Matematikk 1T*. A derived question is whether the expected programming learning objectives are appropriate or not. My subsequent master thesis will investigate this question further. Informed by the findings of this study, a lecture series implementing programming into *Matematikk 1T* will be designed. The design will be tested in a classroom setting during the spring of 2022. The report on the results from the implementation aims to give further insight into which learning outcomes can be expected from pupils enrolled in *Matematikk 1T*.

6 Conclusion

Mathematics is being included in compulsory education across Europe. In Norway, programming has become part of the mathematics subject through the renewal of the National curriculum (LK20) (Sevik, 2016). As described in Forsström and Kaufmann (2018), research on programming in compulsory mathematics education is needed. This study contributes to the field by giving insight into possible approaches to the implementation through lecture designs integrating programming into the subject *Matematikk 1T*.

The constant comparative method has been applied in this project to analyze 21 preliminary projects (PPs) for lecture designs. Descriptive statistics has been a supplementary method of analysis of the data. The PPs are submitted by Norwegian in-service teachers enrolled in programming training at the Norwegian University of Science and Technology (NTNU).

The main finding from this study is three different approaches to integrating programming into *Matematikk 1T* identified through the data material. The approaches can be understood in relation to each other when comparing programming-related student prerequisites and expected learning objectives. Approach 1 builds on the assumption that students have no programming prerequisites, and the learning activities aim to teach students basic programming skills. Approach 2 builds on the assumption that students have obtained necessary programming prerequisites, and hence the learning activities aim to teach students mathematical applications of programming. Approach 3 combines Approach 1 and 2, where both learning new programming concepts and applying programming concepts in mathematics is the aim of the learning activities.

As the study is small-scale and conducted by a single researcher, the findings are not generalizable. Broadening the scope of participants, or doing teacher interviews would give valuable insights to the implementation of programming into *Matematikk 1T*. Additionally, this study calls for further

investigations to give a better understanding of whether the expected student programming skills are appropriate or not. My subsequent master thesis will build on this study, and will investigate the integration of programming into *Matematikk 1T* through implementing a lecture series design in the classroom and reporting on the results.

Bibliography

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. http://hdl.handle.net/11250/2599710

Guzdial, M. (2015). *Learner-centered design of computing education: Research on computing for everyone*. Morgan & Claypool.

Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J. & Szabo, C. (2018). Introductory programming: A systematic literature review. *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, 55–106. https://doi.org/10.1145/3293881.3295779

Meld. St. 28. (2015–2016). *Fag – fordypning – forståelse : En fornyelse av kunnskapsløftet* [Subjects - specialization - understanding: A renewal of *Kunnskapsløftet*]. Kunnskapsdepartementet. https://www.regjeringen.no/contentassets/e8e1f41732ca4a64b003fca213ae663b/no/pdfs/stm201520160028000dddpdfs.pdf

Norstein, A. & Haara, F. O. (2018). *Matematikkundervisning i en digital verden* [Mathematics teaching in a digital world]. Cappelen Damm akademisk.

Norwegian University of Science and Technology. (n.d.). *IT6204 - Anvendt programmering for lærere* [IT6204 - Applied programming for teachers]. Retrieved 4th December 2021, from https://www.ntnu.no/studier/emner/IT6204

NOU 2015:8. (2015). *The school of the future: Renewal of subjects and competences*. Kunnskapsdepartementet. https://www.regjeringen.no/contentassets/da148fec8c4a4ab88daa8b677a700292/en-gb/pdfs/nou201520150008000engpdfs.pdf

Postholm, M. B. (2005). *Kvalitativ metode : En innføring med fokus på fenomenologi, etnografi og kasusstudier* [Qualitative research: An introduction focusing on phenomenology, ethnography and case studies]. Universitetsforlaget.

Robson, C. & McCartan, K. (2016). *Real world research : A resource for users of social research methods in applied settings* (4th ed.). Wiley.

Rouhani, M., Divitini, M. & Olsø, A. (2021). Project-based learning and training of in-service teachers in programming: Projects as a bridge between training and practice. *2021 IEEE Global Engineering Education Conference (EDUCON)*, 262–271. https://doi.org/10.1109/EDUCON46332.2021.9453934

Rouhani, M., Divitini, M., Vujosevic, V., Stai, S. & Olstad, H. A. (2019). Design of a programming course for teachers supporting flexible learning trajectories. *Proceedings of the 8th Computer Science Education Research Conference*, 33–38. https://doi.org/10.1145/3375258.3375263

Sanne, A., Berge, O., Bungum, B., Jørgensen, E. C., Kluge, A., Kristensen, T. E., Mørken, K. M., Svorkmo, A.-G. & Voll, L. O. (2016). *Teknologi og programmering for alle - En faggjennomgang med forslag til endringer i grunnopplæringen - august 2016* [Technology and programming for all - A subject review with proposals for changes in compulsory education - August 2016]. Utdanningsdirektoratet. https://www.udir.no/globalassets/filer/tall-og-forskning/forskningsrapporter/teknologi-og-programmering-for-alle.pdf

Sevik, K. (2016). *Programmering i skolen* [Programming in schools]. Senter for IKT i utdanningen. https://www.udir.no/globalassets/filer/programmering_i_skolen.pdf

The National Committee for Research Ethics in the Social Sciences and the Humanities. (2019). *Guidelines for Research Ethics in the Social Sciences, Humanities, Law and Theology*. https://www.forskningsetikk.no/en/guidelines/social-sciences-humanities-law-and-theology/guidelines-for-research-ethics-in-the-social-sciences-humanities-law-and-theology/

Utdanningsdirektoratet. (2019). *Algoritmisk tenkning* [Computational thinking]. https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/

Utdanningsdirektoratet. (2020). *Curriculum for mathematics vg1 theoretical (mathematics t)* (MAT09-01). https://data.udir.no/kl06/v201906/laereplaner-lk20/MAT09-01.pdf?lang=eng

Wing, J. M. (2010). *Computational thinking–what and why?* http://www.cs.cmu.edu/~CompThink/papers/TheLinkWing.pdf

Appendices

A Preliminary project template

NTNU

Mal v1.0 11.01.2020
Dato 11. januar 2020

IT6204 – Anvendt programmering for lærere

Forprosjekt

Utført av:

Maks 3 deltagere kan jobbe med samme forprosjekt

Tid og sted:

Innhold

Tema på undervisningsopplegg i programmering

Eksempel: Modellering og uttesting av luftmotstand for vannrakett

Undervisningsopplegg i programmering for fag

Eksempel: Matematikk 2P (MAT05-04), Kunst og håndverk (KHV01-02)

Læreplanmål

Eksempel:

Tall og Algebra

Tolke, bearbeide, vurdere og drøfte det matematiske innhaldet i ulike tekstar

Vurdere, velje og bruke matematiske metodar og verktøy til å løyse problem frå ulike fag og samfunnsområde og reflektere over, vurdere og presentere løysingane på ein formålstenleg måte

Rekne formlar, parentesuttrykk og rasjonale og kvadratiske uttrykk med tal og bokstavar.

Omforme ei praktisk problemstilling til ei likning, ein ulikskap eller eit likningssystem, løyse det matematiske problemet både med og utan digitale verktøy, presentere og grunngje løysinga og vurdere gyldigheitsområde og avgrensingar

Geometri

Bruke geometri i planet til å analysere og løyse samansette teoretiske og praktiske problem med lengder, vinklar og areal

Lage og bruke skisser og teikningar til å formulere problemstillingar, i oppgåveløysing og til å presentere og grunngje løysingane, med og utan bruk av digitale verktøy

Funksjonar

Lage, tolke og gjere greie for funksjonar som beskriv praktiske problemstillingar, analysere empiriske funksjonar og finne uttrykk for tilnærma lineære samanhengar, med og utan bruk av digitale verktøy

Gjere greie for funksjonsomgrepet og kunne omsetje mellom ulike representasjonar av funksjonar

Berekne nullpunkt, ekstremalpunkt, skjeringspunkt og gjennomsnittleg vekstfart, finne tilnærma verdiar for momentan vekstfart og gje nokre praktiske tolkingar av desse aspekta

Læringsmål

Eksempel:

Dette opplegget tar utgangspunkt i store deler av nesten alle kompetansemålene for læreplan i 1TY, i tillegg både blokkprogrammering og koding i Python. Følgende læringsmål er foreslått.

Elevene kan gjennomføre en regresjonsanalyse i Geogebra, og vil lære å overføre denne kunnskapen til matlibplot.
Eleven kan ved hjelp av oppgitte formler, omregne fra akselerasjon til både fart og distanse ved hjelp av digitale verktøy.

Elevene kan plotte grafer og bruke rette linjer til å beregne gjennomsnittsfart og momentanfart. Elevene har kjennskap til at farten varierer i alle punkt i en polynomfunksjon. En ball er et fint eksempel på vise at en fart kan være positiv, null og negativ, ved hjelp av rette linjer.
Elevene kan kjenne igjen og forklare lineære og ikke-lineære sammenhenger i praktiske situasjoner og beskrive lineære sammenhenger ved hjelp av matematiske modeller på formen \bigmf(x)=ax+b.

I tillegg skal elevene lære seg å bruke enheter for lengder i modellering i 3D. De skal ha forståelse av volum og kunne legge til og trekke fra ulike volum for å designe figurene slik at micro:bit kan festes til figuren.
Gjennom 3D modelleringen skal eleven lage og bruke skisser og tegninger til å formulere problemstillingene. De skal ha plass til en micro:bit som skal stå fast i selve ballen. De skal og kunne lage presentasjon med og uten digitale verktøy. Dette ville også være god en øvelse til muntlig eksamen i matematikk.

Elevene vil både lære blokkprogrammering i Micro:bit og 3D BlockCads i tillegg til enkle kommandoer i Python for å lage graf i mat.plot.lib, som blir det digitale graf verktøyet.

Forkunnskaper

Eksempel: Be elevene komme med eksempler på forskjellige kurver som viser akselerasjon, fart og avstand. Repeter hva stigningstall i rette linjer kan fortelle om, når de tangerer polynomer. Elevene utfordres i komme med algoritmisk tenking i hvordan vi skal løse programmeringsbiten. Vi skal programmere to micro:bit, en skal måle akselerasjon og en skal skrive ut til et regneark. Elevene utfordres til å tenke på hva akselerasjon er og hvilke opplysninger vi kan bruke fra disse dataene. Elevene vil trenge hjelp til omgjøre akselerasjon til fart, da det ikke er i pensumet i 1TY. Men det er ingenting i veien å forklare sammenhengen

Overordnet prosjektbeskrivelse

Eksempel: Elevene skal sette opp et forsøk som kan vise hvordan luftmotstanden varierer for en vannrakett etter oppskyting og sammenligne teoretiske modellverdier med eksperimentelle verdier. Målingene skal foregå ved hjelp av micro:bit.

Overordnet beskrivelse av undervisningsopplegget

Arbeidsform	*Eksempel:* *Elevene starter med modelleringen, og når 3D printerne jobber, starter elevene på koding.* *For å starte modellering, må de ta fram skyvelære i sitt verktøyskrin. De må måle opp hvor stor mirco:bit de har*
Gjennomføring	*Eksempel:* *Elevene må testkjøre opplegget sitt et par ganger, for å se om de er fornøyd med målingene. Når målingene foreligger, starter databehandlingen og den store matematikk utfordringen. Da skal elevene finne muligheter til å beregne farten og lengden ut fra akselerasjonen…*

Tidsplan

Eksempel:

Del 1: Sette seg inn i micro:bit og bruk av innebygde funksjoner som akselerasjonssensor og radiosender/mottager. Her skal både blokkbasert og tekstbasert programmering brukes. Ca 6 undervisningstimer.

Del 2: …

Utstyrsliste

Eksempel:

3D printere for modellering – klassesett

Micro:bit klassesett på 15 med kabler og batteriholdere.

Egen PC

Gummistrikk klassesett

Blyant og papir

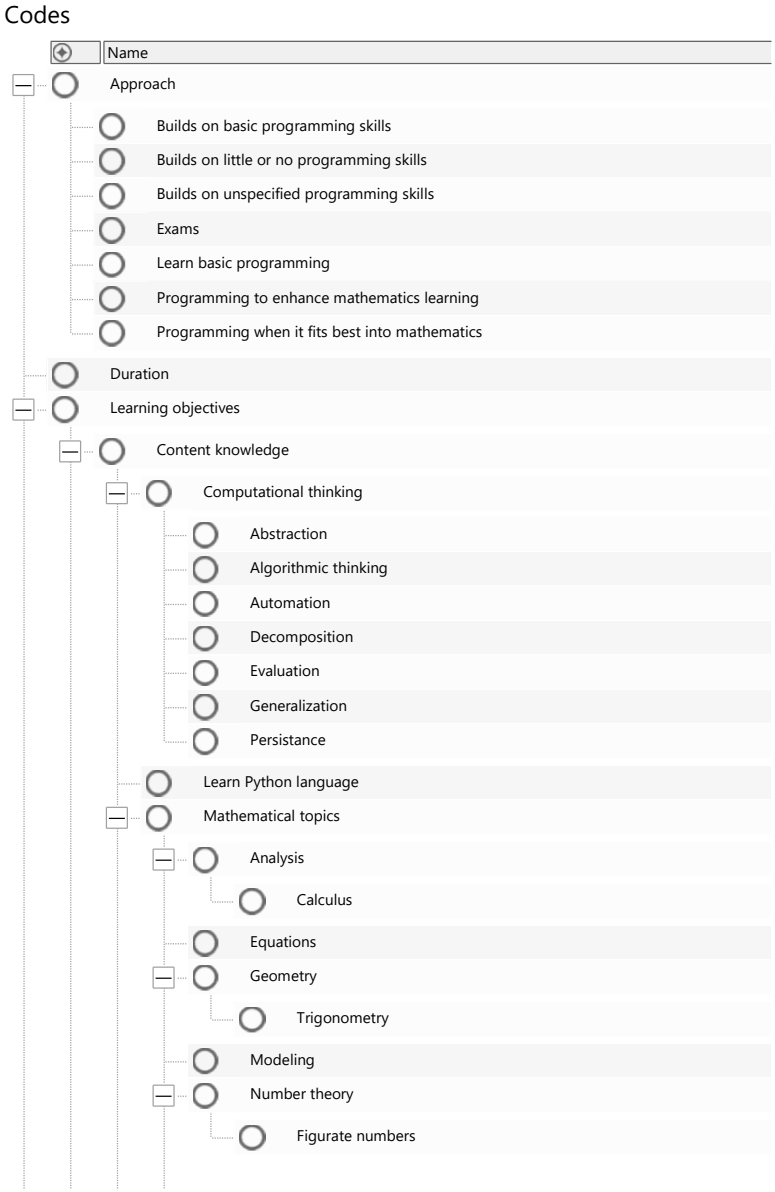
Program:

3D BlocksCad

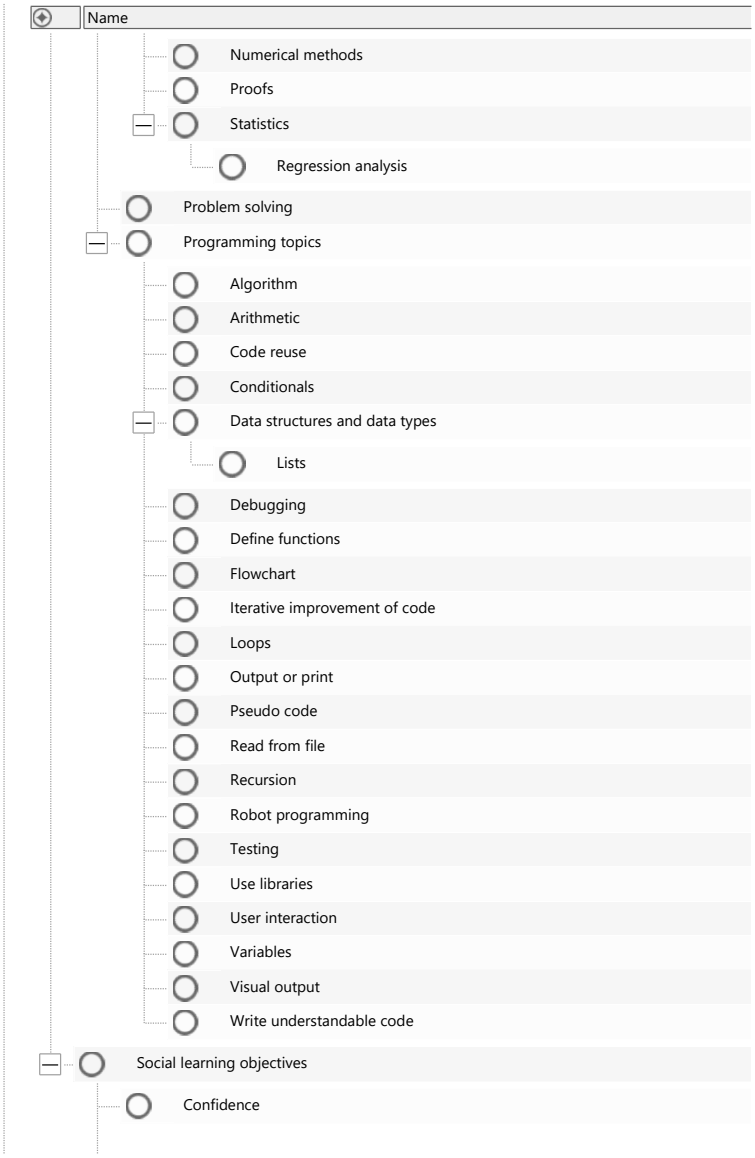
Micro:bit.org

Python og pyplotlib som må være lastet ned før prosjektet starter

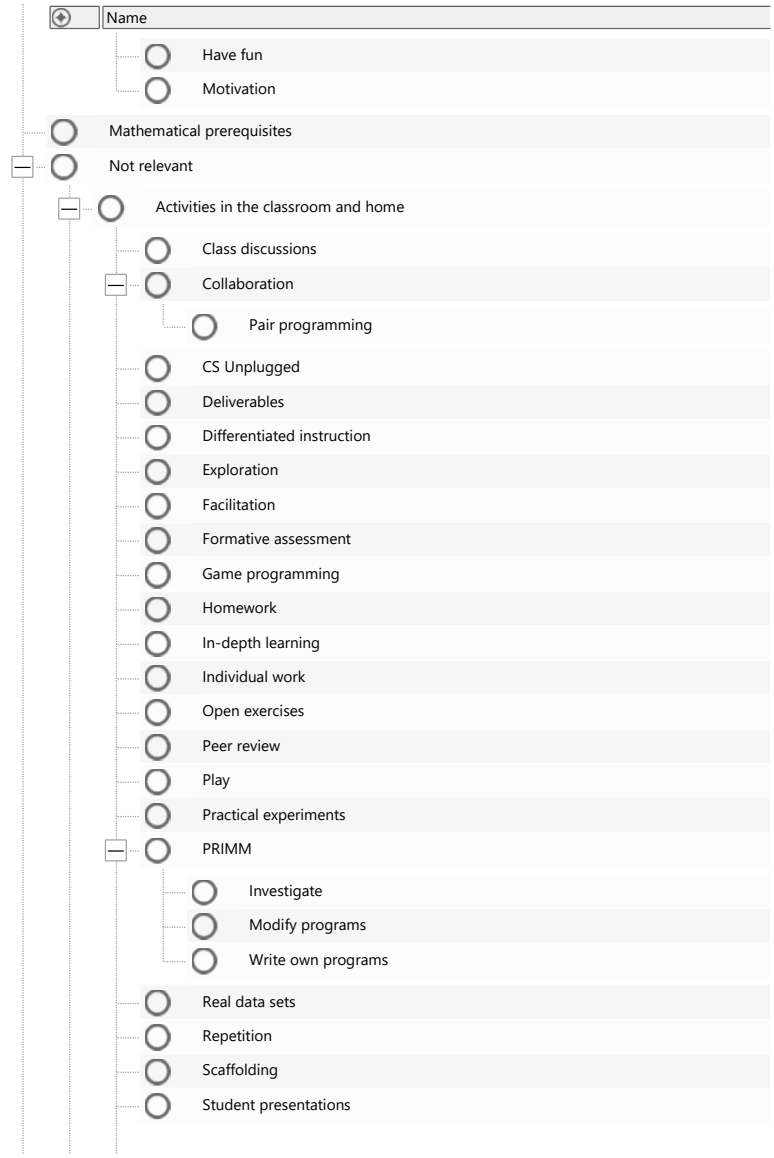
B Coding scheme hierarchy



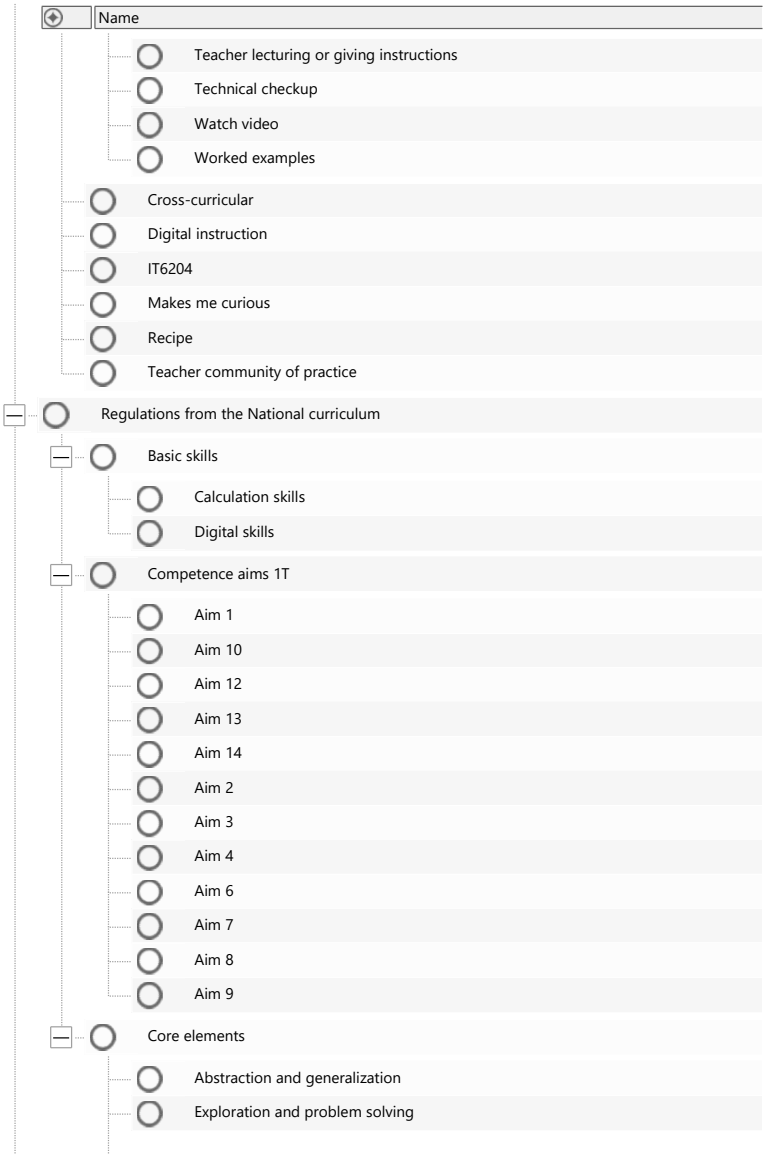
Codes


Name
Numerical methods
Proofs
Statistics
Regression analysis
Problem solving
Programming topics
Algorithm
Arithmetic
Code reuse
Conditionals
Data structures and data types
Lists
Debugging
Define functions
Flowchart
Iterative improvement of code
Loops
Output or print
Pseudo code
Read from file
Recursion
Robot programming
Testing
Use libraries
User interaction
Variables
Visual output
Write understandable code
Social learning objectives
Confidence

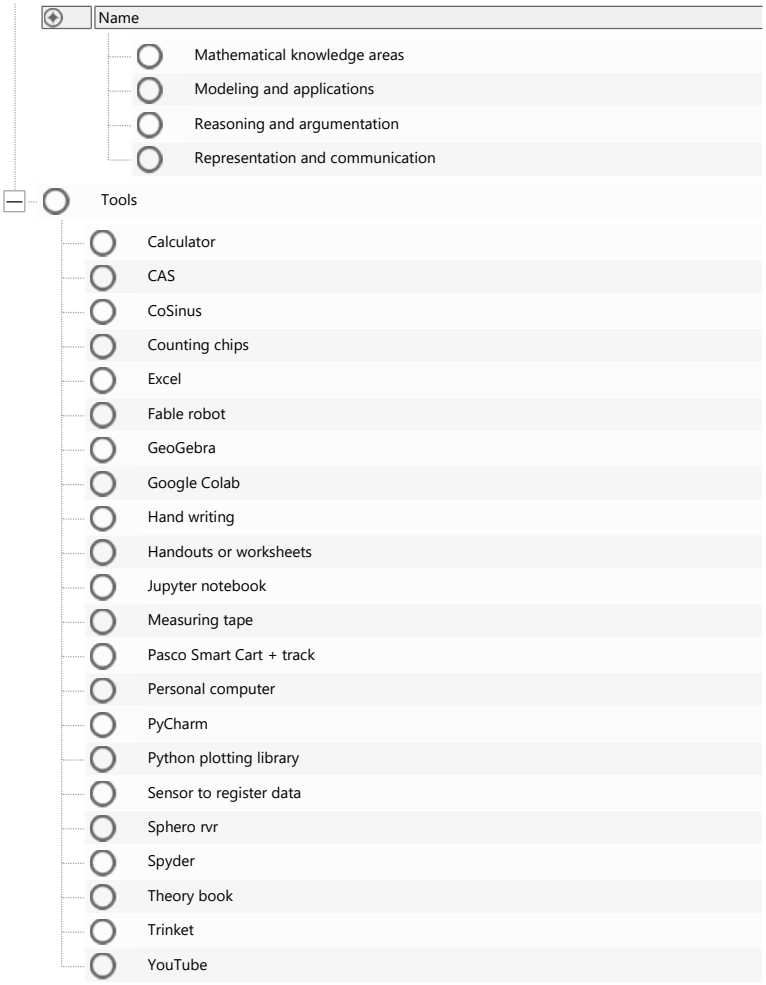
Codes


Name
Have fun
Motivation
Mathematical prerequisites
Not relevant
Activities in the classroom and home
Class discussions
Collaboration
Pair programming
CS Unplugged
Deliverables
Differentiated instruction
Exploration
Facilitation
Formative assessment
Game programming
Homework
In-depth learning
Individual work
Open exercises
Peer review
Play
Practical experiments
PRIMM
Investigate
Modify programs
Write own programs
Real data sets
Repetition
Scaffolding
Student presentations

Codes


Name
Teacher lecturing or giving instructions
Technical checkup
Watch video
Worked examples
Cross-curricular
Digital instruction
IT6204
Makes me curious
Recipe
Teacher community of practice
Regulations from the National curriculum
Basic skills
Calculation skills
Digital skills
Competence aims 1T
Aim 1
Aim 10
Aim 12
Aim 13
Aim 14
Aim 2
Aim 3
Aim 4
Aim 6
Aim 7
Aim 8
Aim 9
Core elements
Abstraction and generalization
Exploration and problem solving

Codes


Name
Mathematical knowledge areas
Modeling and applications
Reasoning and argumentation
Representation and communication
Tools
Calculator
CAS
CoSinus
Counting chips
Excel
Fable robot
GeoGebra
Google Colab
Hand writing
Handouts or worksheets
Jupyter notebook
Measuring tape
Pasco Smart Cart + track
Personal computer
PyCharm
Python plotting library
Sensor to register data
Sphero rvr
Spyder
Theory book
Trinket
YouTube

N Pilot Project Report

12. JANUAR 2021

Programmering i matematikk 1T – Hvordan kan vi støtte matematikklærerne?

RFEL3100 – Høsten 2020

KANDIDATNUMMER: 10018

ANTALL ORD: 4997

Innholdsfortegnelse

1 Innledning

Fagfornyelsen er en oppdatering av læreplanverket, og endrer innholdet i fagene i norsk skole (Meld. St. 28 (2015-2016)). Høsten 2020 trådte de første endringene i kraft for fellesfagene på videregående. I matematikk har programmering blitt en del av fagets metoder, blant annet i matematikk 1T (Utdanningsdirektoratet [Udir], 2020b). Nye kompetansemål er på plass, men hvordan innføringen utspiller seg ser man først nå, gjennom operasjonalisering av læreplanen i klasserommet.

Innføring av programmering på ulike måter i skolefagene er en strømning i Europa (Sevik, 2016). Programmering sees på som en viktig faktor på vei mot et større mål – nemlig å gi elevene undervisning i kompetanser og ferdigheter for det 21. århundre. Det finnes ulike definisjoner på hva disse kompetansene og ferdighetene er (Sevik, 2016). Utredningen «Fremtidens skole» (NOU 2015: 8), er en sentral del av grunnlaget for Fagfornyelsen (Meld. St. 28 (2015-2016)). I rapporten deles kompetanser for det 21. århundre inn i fire områder: Kompetanse i å lære, kompetanse i å utforske og skape, kompetanse i å kommunisere, samhandle og delta, og fagspesifikk kompetanse (NOU 2015: 8). Gjennom kompetansene skal elevene være rustet til å møte morgendagens utfordringer og arbeidsmarked (NOU 2015: 8), og programmering kan være med på å styrke alle de fire kompetanseområdene (Sevik, 2016).

Spesielt potensialet for dypere forståelse er trukket frem som grunnlag for å inkludere programmering i matematikkfaget (Norstein & Haara, 2018). På den andre siden er det argumentert for at programmering vil forsterke stofftrengselen som allerede finnes i matematikkfaget, og dermed hindre at det blir tid til dypere forståelse (Sanne et al., 2016).

Formålet med denne oppgaven er å belyse hvordan man kan møte utfordringen med nettopp stofftrengsel i matematikkfaget. Dette vil jeg gjøre med utgangspunkt i læreren, ettersom det er lærerne som operasjonaliserer læreplanen. Denne oppgaven har derfor til hensikt å besvare dette forskningsspørsmålet:

Hvilke ressurser etterspør tre matematikklærere for å støtte deres arbeid med implementering av programmering i faget matematikk 1T i forbindelse med Fagfornyelsen?

1.1 Oppgavens oppbygning

Først innleder jeg fokusområdet for oppgaven, og dens relevans. Videre definerer jeg sentrale begreper, gir et litteratursammendrag for fagfeltet og forklarer det teoretiske rammeverket

brukt i dataanalysen. Videre beskrives og begrunnes metodene brukt i studien. Del fem presenterer resultater av analysen, og del seks drøfter av resultatene. Del syv er en avslutning.

2 Bakgrunnsteori

2.1 Begrepsavklaringer: Programmering, koding og algoritmisk tenkning

Programmering er et sentralt begrep i denne oppgaven. Det finnes variasjoner i hvor bredt definisjoner av begrepet favner (Norstein & Haara, 2018). Jeg vil i denne oppgaven definere begrepet tett opp mot definisjonen til Sevik (2016). Programmering er en problemløsningsprosess som fører til en programkode som kan kjøres på en datamaskin. En del av prosessen er å forstå og å formulere problemet som skal løses. Programmering favner også feilsøking og forbedring av et program, samt å vurdere ulike løsninger (Sevik, 2016, s. 9).

En distinksjon mellom programmering og koding er verdt å trekke frem. Koding kan forstås som å skrive programkode en datamaskin forstår. Slik begrepene brukes i skolesammenheng, er koding mindre komplekst enn programmering, og begrenses til å skrive mindre avanserte dataprogrammer. Programmering består av blant annet koding, men også av en problemløsningsprosess som gjerne kalles algoritmisk tenkning (Computational thinking) (Norstein & Haara, 2018).

Algoritmisk tenkning er kreativ og systematisk problemløsning. Begrepet forstås i Fagfornyelsen som et sett at arbeidsmåter og nøkkelbegreper i problemløsning. Programmering nevnes ikke i Udirs utgreiing av begrepet, så algoritmisk tenkning behøver ikke å innebære skriving av programkode (Udir, 2019). Likevel er det verdt å presisere at programmering ofte trekkes frem som en aktivitet som fostrer algoritmisk tenkning (Norstein & Haara, 2018; Sevik, 2016). Vedlegg A inneholder oversikt over nøkkelbegreper og arbeidsmåter Udir trekker frem som sentrale i algoritmisk tenkning.

2.2 Litteratursammendrag

Europeiske land har lagt ulike strategier for å innføre programmering i obligatorisk undervisning. Finland og Sverige inkluderer programmering i matematikkfaget i grunnskolen (Sanne et al., 2016; Sevik, 2016). I Estland anses «Teknologi og innovasjon» som et tverrfaglig tema i alle skolefag, og programmering er ett av flere verktøy estiske lærerne kan velge å benytte i fagene sine (Sevik, 2016). I England er «Computing» et eget skolefag, hvor algoritmisk tenkning og programmering er store bestanddeler (Sanne et al., 2016; Sevik,

2016). Oppfordringen i utredningen om fremtidens teknologifag i skolen er å innføre et liknende skolefag i grunnskolen i Norge, «teknologi og programmering» (Sanne et al., 2016).

Tross anbefalingen er det gjennom Fagfornyelsen bestemt at programmering skal inn i eksisterende skolefag. I læreplanen i matematikk 1T står det at elevene skal «formulere og løse problemer ved hjelp av algoritmisk tenkning, ulike problemløsingsstrategier, digitale verktøy og programmering» (Udir, 2020b, s. 5). Utfra ordlyden i kompetansemålet velger jeg derfor i denne oppgaven å se på forskningsfeltet som «anvendelse av programmering i matematikk i videregående opplæring». Forskningsfeltet finner seg i grenseland mellom programmeringsdidaktikk, matematikkdidaktikk og pedagogisk bruk av digitale verktøy. Jeg vil kort presentere status for de tre fagfeltene i Norge i dag.

Problemløsning er trukket frem som en naturlig inngang til programmering i matematikk (Sanne et al., 2016). Problemløsning i matematikk har sin kanon i «How to solve it» av Pólya fra 1945, og også i matematikkdidaktisk forskning har problemløsning en lang tradisjon (Olafsen & Maugesten, 2015). Pedagogisk bruk av digitale verktøy har siden innføring av begrepet «digital kompetanse» med Kunnskapsløftet i 2006 blitt aktuelt (Gilje, 2017). For eksempel SAMR-modellen, som beskriver potensialet teknologi har til å omforme og skape nye muligheter i undervisningsaktiviteter. Fire integrasjonsnivåer av teknologi beskrives i modellen: Uendrede aktiviteter, teknologien gir funksjonelle forbedringer av en uendret aktivitet, modifikasjon av aktiviteter og transformasjon av aktiviteter. (Gilje, 2017). Programmeringsdidaktikk har på den andre siden ikke en like definert tradisjon hverken i norsk eller internasjonal sammenheng. På bakgrunn av strømningene rundt programmering i Europa, er det likevel grunn til å tro at mer forskning vil komme på dette område i nær fremtid (Tiller, 2019).

3 Teoretisk rammeverk – TPACK

TPACK er akronym for Technological PedAgogical Content Knowledge. På norsk finnes det flere ulike oversettelser, men jeg vil forholde meg til «teknologisk-pedagogisk fagkunnskap». TPACK er en utvidelse av Shulmans rammeverk for pedagogisk fagkunnskap (PCK). Opprinnelig ble PCK laget for å forklare hvilke spesielle kunnskaper og ferdigheter som gode lærere har for å undervise et konkret fagstoff på en pedagogisk måte. Shulman ville belyse at en lærers spesialkunnskap er mer enn fagkunnskap og pedagogisk kunnskap sett hver for seg (Shulman, 1986, sitert i Mishra & Koehler, 2006). Hensikten med TPACK er den samme,

men utvider rammeverket til å inkludere påvirkningskraften til anvendelse teknologi i undervisningen (Mishra & Koehler, 2006).

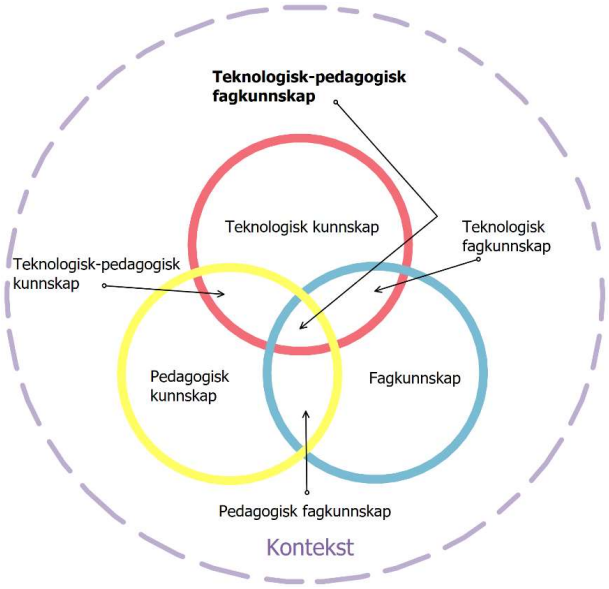
3.1 Syv kunnskapsområder

Grunnmuren i TPACK er de tre kunnskapsområdene teknologisk kunnskap (TK), fagkunnskap (CK) og pedagogisk kunnskap (PK). Satt sammen i et Venn-diagram skaper de fire snitt, hvorpå snittet av alle de tre kunnskapsområdene kalles for TPACK, og betegner lærerens teknologisk-pedagogiske fagkunnskap (Mishra & Koehler, 2006). Rammeverket har fått navn etter dette kunnskapsområdet. De tre områdene illustreres gjerne slik figur 1 på neste side viser. Videre vil TPACK referere til rammeverket, mens teknologisk-pedagogisk fagkunnskap refererer til kunnskapsområdet. Fire av kunnskapsområdene i rammeverket blir beskrevet i denne oppgaven, da det er disse jeg har brukt i koding av transkripsjonene.

Fagkunnskap er lærerens kjennskap til fagstoffet og tema som skal undervises. Teknologisk kunnskap beskriver lærerens kunnskap om og ferdigheter i bruk av alle former for relevante teknologier. Ikke bare digitale teknologier, men også for eksempel tavle og kritt. Teknologisk-pedagogisk fagkunnskap betegner en lærers evne til å velge, og bruke, teknologi til å undervise et spesielt fagstoff på en måte som er hensiktsmessig tatt i betraktning av pedagogiske hensyn som klassetrinn. Det er å gjøre fagdidaktiske avveininger hvor teknologien egner seg til både fagstoffet som er i fokus, og klassen som skal lære (Mishra & Koehler, 2006).

Figur 1

Illustrasjon av TPACK-rammeverket



Note. Ofte illustreres de syv kunnskapsområdene i TPACK-rammeverket slik: Et Venn-diagram med tre parvise og ett trippelt snitt. En kontekst betinger kunnskapen. Bearbeidet fra «Context and Technological Pedagogical Content Knowledge (TPACK): A systematic review», av Rosenberg og Koehler, 2015, *Journal of Research on Technology in Education, 47*(3), s. 187 (https://doi.org/10.1080/15391523.2015.1052663).

3.2 Kontekst i utvidelsen av TPACK

Slik figur 1 viser, er lærerens kunnskap betinget en kontekst i TPACK. Det er fire faktorer som trekkes fram i konteksten: Fagstoffet som skal undervises, klassetrinn, elevforutsetninger og hvilke teknologier som er tilgjengelig (Mishra & Koehler, 2006, s. 1032). Jeg velger i denne oppgaven å bruke en utvidelse av TPACK fra Porras-Hernández og Salinas-Amescua (2013) hvor de nyanserer bestanddelene i konteksten. Dette er fordi funnene i analysen av datamaterialet reflekterer et mer nyansert bilde på kontekst enn det opprinnelige rammeverket legger til rette for.

Porras-Hernández og Salinas-Amescua (2013) forstår konteksten ut fra en omfangsdimensjon (scope) og en aktørdimensjon (actors). Omfangsdimensjonen tar for seg tre nivåer: Makro, meso og mikro, hvor hvert av nivåene inneholder føringer som læreren må forholde seg til.

Dette er på henholdsvis internasjonalt og nasjonalt nivå (makro), i nærmiljøet slik som fylke, kommune og innad på skolen (meso), og i klasserommet (mikro). Aktørdimensjonen beskriver inngående elevers og læreres egenskaper, da disse sees på som de viktigste deltakerne i en undervisningssituasjon. Andre aktører enn lærere og elever inngår på makro-, meso- og mikronivåene (Porras-Hernández & Salinas-Amescua, 2013).

3.2.1 Omfangsdimensjonen

Styringsdokumenter på nasjonalt nivå faller innenfor makronivået (Porras-Hernández & Salinas-Amescua, 2013). I norsk sammenheng forstår jeg dette nivået som blant annet føringer fra Udir og Kunnskapsdepartementet, slik som læreplaner, eksamensordning, samt fag- og timefordeling. Videre inneholder makronivået informasjon om andre sosiale, politiske, økonomiske og teknologiske forhold på både nasjonalt og globalt nivå (Porras-Hernández & Salinas-Amescua, 2013), slik som for eksempel strømninger i Europa rundt kompetanser for det 21. århundre.

3.2.2 Aktørdimensjonen – Elever

Læreren må ha begrep om egenskapene til elevene i klassen (learners' inner characteristics). Dette handler blant annet om deres holdninger og forkunnskaper, samt forforståelser om seg selv, andre mennesker og faginnhold. Videre er elevene formet av kulturen de er en del av (habitus), og læreren må derfor ha kunnskap også om konteksten rundt elevene. Elevene både som gruppe og enkeltindivider er med andre ord faktorer som påvirker valgene læreren gjør, og har mulighet til å gjøre i undervisningen (Porras-Hernández & Salinas-Amescua, 2013, s. 231). Også når de skal velge å ta i bruk programmering, som er spesielt relevant for denne oppgaven.

Da meso- og mikronivåene i omfangsdimensjonen og lærerens egenskaper i aktørdimensjonen ikke benyttes i analyse av datamateriale i denne oppgaven, vil jeg ikke beskrive dem mer inngående.

4 Metode

4.1 Forskningsdesign

Studien jeg har gjennomført hører til det kvalitative forskningsparadigme. Dette kan begrunnes gjennom formål og metoder. Studien har ikke til hensikt å tallfeste generelle tendenser, men forsøker heller gjennom et fåtall intervjuer å forstå en kontekstbetinget situasjon. Like viktig er det at designet er fleksibelt, og dermed har blitt til i løpet av

semesteret (Robson & McCartan, 2016). En sentral del av konteksten for studien er at dette er første semester hvor programmering er en del av faginnholdet i matematikk, og da spesielt i matematikk 1T som jeg har undersøkt. Dermed har jeg lagt opp studien til å være småskala med trekk fra casestudie. Det er *fenomenet* «innføring av programmering i matematikk 1T» som studeres (Robson & McCartan, 2016).

4.2 Datainnsamling

For å belyse forskningsspørsmålet har jeg gjennomført semistrukturerte intervjuer med lærere fra to ulike fylker som alle underviser faget matematikk 1T høsten 2020. På forhånd laget jeg en intervjuguide, se vedlegg B. Fordi intervjuene var semistrukturerte, varierte ordlyden og rekkefølgen noe i alle intervjuene (Robson & McCartan, 2016). Intervjuene skulle opprinnelig gjennomføres på skolen hvor lærerne arbeidet, men på grunn av smittesituasjonen knyttet til covid-19, ble intervjuene flyttet over på videokonferanseverktøyet Zoom. Vendingen muliggjorde intervjuer med lærere også i en annen del av landet. Fire intervjuer ble gjennomført, men jeg har valgt å se bort fra ett av intervjuene. Materialet jeg satt igjen med etter datainnsamlingen var lydopptak av intervjuene, samt håndskrevne notater jeg tok underveis.

4.3 Analysemetode

I denne oppgaven har jeg benyttet en tilpasning av konstant komparativ analysemetode (CCM). CCM har sitt opphav i *Grounded theory*, en rigorøs metode for datadreven kvalitativ forskning, men egner seg også i andre fleksible forskningsdesign som casestudier (Postholm, 2005). I CCM deles analyseprosessen inn i tre faser: Åpen koding, aksial koding og selektiv koding. Målet er å merke liknende meningsbærende datasegmenter med *koder*, gruppere kodene i *kategorier*, og abstrahere kategoriene til en eller flere *kjernekategorier* som favner essensen av budskapet i datamaterialet (Postholm, 2005; Robson & McCartan, 2016). Jeg valgte CCM fordi jeg ikke visste helt hvilket teoretisk rammeverk som best ville forklare funnene mine, og da kunne en datadreven tilnærming passe. Jeg har tilpasset metoden ved at jeg ikke har identifisert en kjernekategori gjennom selektiv koding, men besvarer forskningsspørsmålet gjennom tre kategorier fra den aksiale kodingsprosessen.

4.3.1 Forarbeid

Ifølge Robson og McCartan (2016) er den første delen av dataanalyse å bli kjent med datamaterialet. Jeg laget kronologiske innholdsoversikter med tidsstempel som kartla ordlyden og rekkefølgen av spørsmål som jeg stilte, samt kulepunkter i stikkordsform over

svarene til lærerne. I margen av innholdsoversikten noterte jeg interessante utsagn, en metode som kalles «memoing» (Robson & McCartan, 2016). Dette la jeg grunnlaget for å velge ut hvilke deler av intervjuene som inneholdt de mest interessante utsagnene, og belyste likheter og ulikheter i oppfatningene til lærerne jeg intervjuet. Lærerne ble anonymiserte, og fikk de fiktive navnene Helge, Kristian og Pernille. Deretter transkriberte jeg deler av intervjuene. Dette tok meg over i neste fase.

4.3.2 Åpen koding

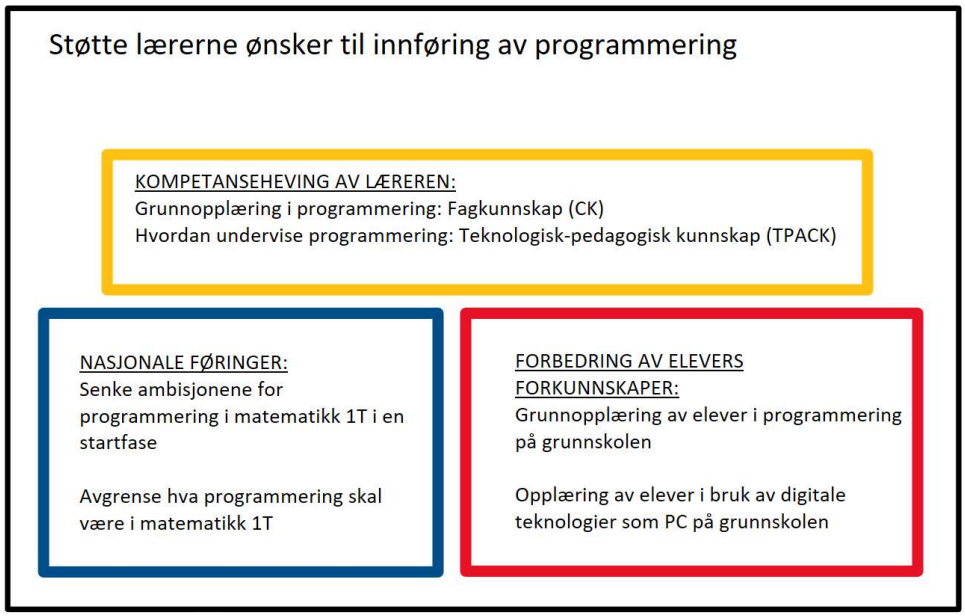
Problemer med innføringen av programmering og tilknyttede støttetiltak ble tidlig to gjennomgående tema i intervjuene. I den åpne kodingsprosessen markerte jeg relaterte utsagn fra transkripsjonene med fargekoder, som til slutt ble til seks ulike koder. Kodene gjengis i figur 2. Det var en iterativ prosess mellom transkripsjonene og ulike teoretiske rammeverk, før jeg identifiserte at TPACK-rammeverket med en nyansering av kontekst favnet om de fargekodene utsagnene.

4.3.3 Aksial koding

Den åpne kodingsprosessen førte til en glidende overgang til aksial koding. Tre kategorier av koder utkrystalliserte seg i datamaterialet: Lærerens ønske om kompetanseheving, forbedring av elevers forkunnskaper, og nasjonale føringer. De tre kategoriene bruker jeg som basis for å besvare forskningsspørsmålet. Figur 2 viser hvilke koder jeg har brukt, samt innunder hvilke kategorier kodene hører til.

Figur 2:

Koder og kategorier



Note. Relasjonen mellom koder og kategorier jeg endte opp med etter konstant komparativ analyse av datamaterialet. Egen illustrasjon.

4.4 Refleksjoner rundt metode

Jeg har i dette emnet lært mer og mer om forskningsdesign underveis, og følgelig har designet utkrystallisert seg i takt med læringsprosessen. Jeg hadde tidlig et for bredt forskningsspørsmål, noe som ble reflektert i intervjuene. Det hadde det vært mer fruktbart å planlegge en snevrere studie fra starten av. Dette er en svakhet ved min forskning, ettersom det førte til at jeg ikke hadde en klar nok sammenhengen mellom det endelige forskningsspørsmålet og datamaterialet jeg samlet inn.

5 Analyse

5.1 Epistemologisk analyse

Programmering er nytt i læreplanen, og det er derfor fortsatt uklart akkurat hva som utgjør kjernen av «programmering i matematikk 1T». Jeg tolker formuleringen i kompetansemålet dithen at man skal forbi grunnopplæring, og ta steget opp til *anvendelse* av programmering i

matematikkfaget. Jeg vil i denne delen av oppgaven redegjøre for valg av teknologi som må tas av læreren, samt de grunnleggende programmeringsverktøyene som elevene bør å kjenne til for å anvende programmering. Materialet jeg bygger denne analysen på er læringsressurser for elever og lærere. Se vedlegg C.

5.1.1 Valg av teknologi

For å skrive programkode, må man ha valgt et programmeringsmedium og et programmeringsspråk. Normen ser ut til å bli tekstbasert programmering i Python 3 i matematikk 1T. Videre vil jeg referere til Python 3 når jeg skriver Python. Alle ressurser jeg har undersøkt bruker Python. Også den opprinnelige utgaven av eksempeloppgaver til eksamen fra Udir benyttet Python, selv om de nå har trukket tilbake valg av programmeringsspråk (Udir, 2020a). At valget har falt på Python ser ut til å være noe tilfeldig, og noe faglig forankret. Et hovedpoeng for å velge Python til utdanningsformål er at språket har lav inngangsterskel og høyt tak. Begrunnelsene for dette kan oppsummeres som nedenfor:

- Språket er godt dokumentert åpent på nettet.
- Python er mye brukt i arbeidslivet.
- Syntaksen ligger tett opp mot de engelske språket.
- Python er tilgjengelig gratis for de fleste operativsystem.
- Det finnes utallige utvidelser til Python.

(Tollervey, 2015)

I tillegg til å velge programmeringsspråk og programmeringsmedium, må man velge utviklingsmiljø. Dette består gjerne av en kodeeditor og en tolker (interpreter), slik at man både kan skrive og utføre koden. De to utviklingsmiljøene som virker mest utbredt for bruk sammen med Python er Spyder i Anaconda, og Trinket. Trinket er en nettbasert tjeneste, mens Spyder laster man ned lokalt på datamaskinen sin.

Begge de to alternativene håndterer nødvendige utvidelser av Python som trengs i matematikk 1T. Den automatiske håndteringen ser ut til å være en viktig grunn for valg av utviklingsmiljø, da alternativet til automatisk håndtering av utvidelser er å selv installere de nødvendige utvidelsene, noe som krever en viss teknisk kompetanse.

5.1.2 Grunnleggende programmering

Da det ville blitt et eget innføringskurs å forklare elementene i grunnleggende Python-programmering, vil jeg heller forsøke å gi oversikt over det mest gjentakende innholdet i materialet jeg har gjennomgått. Oversikten er ikke utfyllende, og variasjoner forekommer.

Krever konseptuell forståelse, og kjennskap til syntaks:

- Variabler og datatyper
- Aritmetiske og logiske operatorer
- For-løkker og while-løkker
- If-elif-else-strukturen for kontroll av programflyt
- Input og output gjennom de innebygde funksjonene «input» og «print»
- Formattering av store og små tall
- Definering av og kall til egne funksjoner
- Importering og bruk av utvidelser

Retningslinjer for god programmeringskultur:

- Kommentering av kode
- Feilsøking og lesing av feilmeldinger

5.2 Resultater

5.2.1 – Kompetanseheving av lærerne

Kompetanseheving av lærerne er en nødvendig ressurs for å støtte innføringen av programmering i matematikk 1T. Dette uttrykker både Helge som har arbeidet mange år i skolen som lærer i naturfag, fysikk og matematikk, og Kristian som nettopp fullførte lektorutdanningen i biologi og matematikk. De ønsker likevel ikke den samme kompetansehevingen, de har ulike behov.

Helge uttrykker at den viktigste støtten for han vil være et kompetanseløft i programmering. Jeg har plassert følgende tilknyttede utsagn inn i koden fagkunnskap (CK):

> H: For min del har jeg ønske om mest å lære meg skikkelig å bruke programmeringen selv. Med syntaks, og bruke kommandoene, og bli fortrolig med kommandoene. […] [D]u må jo egentlig ha det i hodet selv, og ha jobbet med det selv, og ha programmert *selv* en del oppgaver. Og gjort feilene selv. Det er et langt lerret å bleke det å skulle

> *undervise et fag*. Når du knapt nok har hatt det før! For min del, har jeg hatt det for 30 år siden.

Kristian opplever at han gjennom blant annet numeriske matematikkfag i lektorutdanningen har fått tilstrekkelig erfaring med programmering i Python. Jeg koder dette som at han er tilfreds med nivået på egen fagkunnskap om programmering. Noe han lurer på er likevel hvordan han skal undervise programmering i klasserommet. Dette har jeg plassert i koden teknologisk-pedagogisk fagkunnskap (TPACK).

> K: Mange av de lærerne som jobber på skolen vår, de er bare trent med å programmere. Sånn som jeg også er. Vi har hatt veldig lite om hvordan man skal *undervise* elevene. Det må du finne litt ut av selv. Jeg skulle gjerne ønske man hadde didaktikkurs i forhold til undervisning av elevene.

At to lærere med såpass forskjellige bakgrunner trekker frem ønske om et kompetanseløft kan bety at det er grunn til å prioritere etterutdanning for matematikklærere med innføringen av de nye læreplanene.

5.2.2 – Nasjonale føringer som tar hensyn til elevenes forkunnskaper

De tre lærerne i denne studien trekker fram at egen kompetanseheving ikke er nok støtte for å implementere programmering i matematikk 1T på en god måte. De savner tydeligere føringer for faginnholdet på nasjonalt nivå, i kombinasjon med at det tas høyde for at elevenes forkunnskaper i programmering øker over tid. Dette forstår jeg som støtte til omfangsdimensjonen og aktørdimensjonen, som i utvidelsen av TPACK-rammeverket er betingende for lærerens teknologisk-pedagogiske fagkunnskap, og avgjørende for hvordan og når lærere velger å ta i bruk teknologier (Porras-Hernández & Salinas-Amescua, 2013).

Omfangsdimensjonen: Føringer på makronivå

Lærerne ønsker en avklaring av hvilken del av programmeringsopplæringen de skal ha ansvar for i matematikk 1T. Helge problematiserer planlegging av undervisning når programmeringsopplæring kan bety mye forskjellig:

> H: [...] Hvert fall nå i en startperiode hadde det vært greit å ha noen føringer for: Hvor mange uker er det meningen at vi skal bruke på [programmering] [i] faget? Hva skal vi legge vekt på? Skal vi legge vekt på syntaksen? Eller skal vi *bare* tenke på forståelsen for koden og kommandoene, er det det som er hovedvekten? Det er *lite* føringer i de

> ulike fagene. Det blir veldig individuelt: Hvor mye læreren legger vekt på programmering. Og de ulike *delene* av programmeringen.

Han utdyper videre at det må defineres et ambisjonsnivå for de ulike delene av programmeringsopplæringen:

> H: Hvor høye ambisjoner har man? Noen har kanskje bare ambisjoner om at hvis elevene klarer å bruke print-funksjonen og lage helt enkle løkker, så [er det nok]. Mens andre vil at de skal lage programmer på både én og to sider, og skal gjøre avanserte gruppearbeid.

Pernille er den eneste av de tre lærerne som allerede har etterutdanning i programmering. Hun oppsummerer hvor viktig avklaringer vil være for hennes undervisning:

> P: Hvis jeg vet ved skolestart hva som er målet, hvor [langt] jeg skal komme, så ville jeg sikkert klare med mitt nivå og min erfaring å lage en plan for å komme til dette målet.

Aktørdimensjonen: Elevenes forkunnskaper

At lærerne ønsker seg følgende avklaringer, henger tett sammen med elevene de skal undervise. Elevenes forkunnskaper må tas hensyn til i undervisningen. De fleste elevene i deres klasser har aldri vært borti programmering før. Elevenes evner til å håndtere digitale teknologier som PC er også på et lavt nivå. Lærerne understreker at ambisjonsnivået i faget må henge sammen med elevenes forkunnskaper. Entydig uttrykker de tre lærerne at de fleste elevene ikke har tilstrekkelige forkunnskaper til at *anvendelse* av programmering kan forventes allerede våren 2021. Kristian benytter eksempeleksamensoppgavene som en rettesnor for ambisjonsnivået til programmering i læreplanen:

> K: […] [J]eg synes at de gikk veldig *hardt* på. Og hvis det er dette nivået elevene skal kunne etter bare *ett* år med programmering, så har den norske skolen et problem.

Han utdyper videre hva han faktisk bruker tiden på i undervisningen:

> K: […] Jeg skulle gjerne ønske at alle [elevene] var kjempegode i det digitale, og det tekniske rundt, slik at man kunne fokusere på den programmeringsbiten. Men det blir mye mer komplekst enn det når man får det tekniske rundt i tillegg.

Læreren Helge uttrykker noe av den samme frustrasjonen:

> H: […] Det skjer oftere enn man tror altså! […] Så det er litt sånn idealisert føler jeg, hele greia med at vi skal putte inn ekstra ting i fagene som ikke var der *før*. […] For to pluss to sånn tidsmessig, blir ikke bestandig fire: To pluss to kan bli både fem og seks og sju, sånn tidsmessig, når man putter inn ekstra momenter. Det føler jeg ikke er helt tatt hånd om.

Det Helge og Kristian trekker frem, er utfordringer jeg har plassert inn under kodene *grunnopplæring i programmering* og *opplæring i bruk av datamaskin*. Dette er aktiviteter klassen bruker tid på i opplæringen som ikke bidrar direkte inn mot læring i kompetansemålene. Det er et for lavt nivå til å kunne forstås som *anvendelse* av programmering i matematikk. De bruker dermed av tiden som skulle vært brukt til faget matematikk 1T, til heller å heve elevenes kompetanse opp til nivået som skulle vært forkunnskaper i tråd med den nye læreplanen.

6 Diskusjon

Den utvidede versjonen av TPACK-rammeverket som jeg har brukt i denne oppgaven nyanserer at lærerens kunnskap er kompleks – og hvordan ulike lærere med ulike bakgrunner ønsker ulik kompetanseheving. Rammeverket tar høyde for at den teknologisk-pedagogiske kunnskapen til læreren er betinget av en kontekst bestående av egenskapene til både læreren og elevene, samt føringer på makro- meso- og mikronivå. Gjennom rammeverket blir mangelen på føringer på nasjonalt nivå konkretisert som et problemområde i makrodimensjonen. Videre gir rammeverket et teoretisk forankret grunnlag for å problematisere at ny teknologi, slik som programmering i Python, har stor innvirkning på lærerens teknologisk-pedagogiske kunnskap. Rammeverket har dermed gitt en kompleks og detaljert inngang til å besvare forskningsspørsmålet:

Hvilke ressurser etterspør tre matematikklærere for å støtte deres arbeid med implementering av programmering i faget matematikk 1T i forbindelse med Fagfornyelsen?

Oppgaven begrunner at kompetanseheving må tilbys for alle lærere som skal undervise i matematikk 1T. Læreren bør ha medbestemmelsesrett i akkurat hva de ønsker å lære mer om, da de sitter med ulik kunnskap fra arbeidsliv og utdanning. Jakobsen (2019) kommer i sin masteroppgave til en liknende konklusjon, og peker på at etterutdanning kan bidra til at lærerne har større forståelse for hvilken del av programmeringen som er mest relevant å undervise i deres matematikkfag. Slik læreren Helge sa: «Det er et langt lerret å bleke, å

skulle undervise et fag når du knapt har hatt det selv!». Kompetanseheving av lærere må tas på alvor, og etterutdanning er en mulig løsning.

Potensialet for programmering er stort. Det kan bidra til å utvikle elevenes kompetanse i å kommunisere, samhandle og delta, kompetanse i å utvikle og skape, fagspesifikk kompetanse, og kompetanse i å lære (Sevik, 2016). Dette er kunnskapene og ferdighetene som er forespeilet at elevene vil trenge i samfunnet de skal vokse opp i (NOU 2015: 8). Denne oppgaven bidrar til å belyse hvordan visjonen kan realiseres gjennom å sette inn tre ressurser, i stedet for at stofftrengsel blir utfallet av programmering i matematikkfaget. I tillegg til at lærerne i matematikk 1T ønsker seg et kompetanseløft, etterspør de at elevene må lære grunnleggende programmering på grunnskolen.

Progresjonen i kompetansemålene reflekterer at elevene *skal* få grunnleggende opplæring i programmering på grunnskolen (Udir, 2020b). At dette gjennomføres vil ha stor betydning for matematikk 1T i årene som kommer. Kompetansemålet som spesifikt nevner programmering i matematikk 1T, har en ordlyd som vektlegger at programmering skal brukes som verktøy i problemløsning (Udir, 2020b). Dette er en anvendelse av programmering, og forutsetter at elevene er kjent med hvilke grunnleggende strukturer som er tilgjengelige for dem i programmering. Denne studien belyser at for å ha tid til nettopp å *anvende* programmering på VG1, må grunnskolen ta dette ansvaret. I kombinasjon med tydeligere avgrensning på nasjonalt nivå av matematikk 1T sitt kortsiktige og langsiktige ansvar i programmeringsopplæringen, vil stofftrengselen da kunne motvirkes.

7 Avslutning

I denne oppgaven har jeg undersøkt hvilke ressurser lærere selv oppfatter som hensiktsmessige for å gjøre innførelsen av det nye læreplanverket smidigere. Studien har jeg gjort med formålet å forstå, fra perspektivet til de lærerne som står for undervisningen i klasserommet, hvordan man kan møte utfordringene som oppstår i forbindelse med innføring av programmering i faget. For å besvare forskningsspørsmålet har jeg gjennomført analyse av intervjuer med tre lærere som alle har ulik utdanning og fartstid i læreryrket.

Et av de viktigste funnene mine er at lærerens kompetanse trenger et løft. Dette kan gjerne skje gjennom etterutdanning. Lærere har individuelle behov basert på hva de allerede mestrer, og det er derfor naturlig å trekke slutningen at ett og samme etterutdanningsløp ikke nødvendigvis vil passe alle lærere. Det andre viktige funnet er at heving av lærerens kompetanse alene ikke er nok. Målet for programmeringsopplæringen i faget er ikke tydelig nok avgrenset, og ambisjonene er foreløpig for høye når elevenes forkunnskaper tas i betraktning. Spesielt at læreren Pernille som har etterutdanning sier dette, veier tungt. Grunnopplæring i programmering på grunnskolen, og føringer på nasjonalt nivå er derfor også etterspurte ressurser.

Interessant for videre forskning er elevperspektivet. Hva er elevenes opplevelse av programmering i matematikk 1T? Hva sitter de igjen med etter noen år med nye læreplaner i matematikk? Jeg vil ta elevperspektivet med meg videre i masterskriving og arbeidsliv. Det viktigste jeg tar med meg fra denne studien er at et godt forskningsdesign er grunnmuren i et prosjekt. Problemene jeg har hatt med å definere forskningsspørsmålet på en presis måte kan spores tilbake til for lite grundig forarbeid med forskningsdesignet. Men jeg har lært, og er fornøyd med at jeg etter RFEL3100 er mer beredt til å skrive masteroppgave.

Referanser

Gilje, Ø. (2017). *Læremidler og arbeidsformer i den digitale skolen*. Fagbokforlaget.

Jakobsen, R. E. (2019). *Programmering i matematikk – muligheter og utfordringer : En studie rundt innføringen av programmering som del av matematikkfaget i den norske skolen og hvilke argumenter som taler for eller imot innføringen* [Masteroppgave, Universitetet i Agder]. AURA. https://uia.brage.unit.no/uia-xmlui/handle/11250/2646381

Meld. St. 28 (2015-2016). *Fag – fordypning – forståelse : En fornyelse av Kunnskapsløftet*. Kunnskapsdepartementet. https://www.regjeringen.no/contentassets/e8e1f41732ca4a64b003fca213ae663b/no/pdfs/stm201520160028000dddpdfs.pdf

Mishra, P. & Koehler, M. J. (2006). Technological pedagogical content knowledge: A framework for teacher knowledge. *Teachers College Record*, *108*(6), 1017-1054. https://doi.org/10.1111/j.1467-9620.2006.00684.x

Norstein, A. & Haara, F. O. (2018). *Matematikkundervisning i en digital verden*. Cappelen Damm.

NOU 2015: 8. (2015). *Fremtidens skole: Fornyelse av fag og kompetanser*. Kunnskapsdepartementet. https://www.regjeringen.no/contentassets/da148fec8c4a4ab88daa8b677a700292/no/pdfs/nou201520150008000dddpdfs.pdf

Olafsen, A. R. & Maugesten, M. (2015). *Matematikkdidaktikk i klasserommet* (2. utg.). Universitetsforlaget.

Porras-Hernández, L. H. & Salinas-Amescua, B. (2013). Strengthening TPACK: A broader notion of context and the use of teacher's narratives to reveal knowledge construction. *Journal of Educational Computing Research*, *48*(2), 223-244. https://doi.org/10.2190/EC.48.2.f

Postholm, M. B. (2005). *Kvalitativ metode : En innføring med fokus på fenomenologi, etnografi og kasusstudier*. Universitetsforlaget.

Robson, C. & McCartan, K. (2016). *Real world research : A resource for users of social research methods in applied settings* (4. utg.). Wiley.

Rosenberg, J. M. & Koehler, M. J. (2015). Context and technological pedagogical content knowledge (TPACK): A systematic review. *Journal of Research on Technology in Education*, *47*(3), 186-210. https://doi.org/10.1080/15391523.2015.1052663

Sanne, A., Berge, O., Bungum, B., Jørgensen, E. C., Kluge, A., Kristensen, T. E., Mørken, K. M., Svorkmo, A.-G. & Voll, L. O. (2016). *Teknologi og programmering for alle - En faggjennomgang med forslag til endringer i grunnopplæringen - august 2016*. Utdanningsdirektoratet. https://www.udir.no/globalassets/filer/tall-og-forskning/forskningsrapporter/teknologi-og-programmering-for-alle.pdf

Sevik, K. (2016). *Programmering i skolen*. Senter for IKT i utdanningen. https://www.udir.no/globalassets/filer/programmering_i_skolen.pdf

Tiller, M. (2019). *Didaktiske valg i valgfag programmering* [Masteroppgave, OsloMet]. ODA. https://oda.oslomet.no/handle/10642/8735

Tollervey, N. H. (2015). *Python in education: Teach, learn, program*. O'Reilly.

Utdanningsdirektoratet. (2019). *Algoritmisk tenkning*. https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/

Utdanningsdirektoratet. (2020a). *Eksempeloppgaver i matematikk T*. Hentet 22. desember 2020 fra https://www.udir.no/eksamen-og-prover/eksamen/eksempeloppgaver/matematikk-eksempeloppgaver/

Utdanningsdirektoratet. (2020b). *Læreplan i matematikk fellesfag Vg1 teoretisk (matematikk T)* (MAT09-01). https://data.udir.no/kl06/v201906/laereplaner-lk20/MAT09-01.pdf?lang=nob

Vedlegg A – Plakat om algoritmisk tenkning fra Udir

Figur 1

Den algoritmiske tenkeren

Note. Nøkkelbegreper og arbeidsmåter som kjennetegner den algoritmiske tenkeren. Fra *Algoritmisk tenkning,* av Utdanningsdirektoratet, 2019 (https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/).

Vedlegg B – Intervjuguide innsendt til Norsk senter for forskningsdata (NSD)

Intervjuguide

Har du brukt programmering i din(e) klasse(r) denne høsten?

Hvilke muligheter for læring ser du i innføringen av programmering i faget?

Hvilke utfordringer forventer du å støte på i forbindelse med programmering?

Hvilke krav føler du det stilles til deg som lærer i forbindelse med innføring av programmering?

Hvordan har du forberedt deg til å undervise programmering denne høsten?

Om noen, hvilken oppfølging ønsker du for å være mer forberedt til å undervise i programmering?

Vedlegg C – Grunnlag for epistemologisk analyse

Nye lærebøker i matematikk 1T

Matematikk 1T

Borge, I. C., Engeseth, J., Haug, H., Heir, O., Moe, H., Norderhaug, T. T., & Vie, S. M. (2020). *Matematikk 1T* (4. utgave). Aschehoug undervisning.

Mønster

Kalvø, T., Opdahl, J. C. L., Skrindo, K., Weider, Øystein J., & Wilmann, S. (2020). *Mønster : matematikk 1T : studieforberedende utdanningsprogram*. Gyldendal.

Sinus 1T

Oldervoll, T., Svorstøl, O., Vestergaard, B., Gustafsson, E., Osnes, E. R., Pedersen, T. A., & Jacobsen, R. B. (2020). *Sinus 1T : matematikk : studieforberedende vg1* (4. utgave). Cappelen Damm.

Programmeringsbøker myntet på lærere

Bueie, H. (2019). *Programmering for matematikklærere*. Universitetsforlaget.

Haraldsrud, A. D., Sveinsson, H. A. & Løvold, H. H. (2020). *Programmering i skolen.* Universitetsforlaget.

Tollervey, N. H. (2015). *Python in education: Teach, learn, program*. O'Reilly.

Eksempeleksamensoppgaver fra Utdanningsdirektoratet

Utdanningsdirektoratet. (2020). *Eksempeloppgaver i matematikk T*. Hentet 22. desember 2020 fra https://www.udir.no/eksamen-og-prover/eksamen/eksempeloppgaver/matematikk-eksempeloppgaver/

Anne M. V. Bosch

Art Production with Programming and Trigonometry

NTNU
Kunnskap for en bedre verden