Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for samfunns- og utdanningsvitenskap
Institutt for lærerutdanning

Siv Andrea Gulaker

Matematikkundervsining i programmeringsmiljøet Scratch

En kvalitativ casestudie av et utvalg 10. klasse elevers arbeid i Scratch

Masteroppgave i matematikkdidaktikk 5.-10.
Veileder: Hermund André Torkildsen
Medveileder: Marit Buset Langfeldt
Juni 2022

NTNU
Kunnskap for en bedre verden

Siv Andrea Gulaker

Matematikkundervsining i programmeringsmiljøet Scratch

En kvalitativ casestudie av et utvalg 10. klasse elevers arbeid i Scratch

Masteroppgave i matematikkdidaktikk 5.-10.
Veileder: Hermund André Torkildsen
Medveileder: Marit Buset Langfeldt
Juni 2022

Norges teknisk-naturvitenskapelige universitet
Fakultet for samfunns- og utdanningsvitenskap
Institutt for lærerutdanning

Sammendrag

Programmering og algoritmisk tenkning er blitt en del av den norske læreplanen (Utdanningsdirektoratet, 2020), og matematikkfaget har særlig ansvar for å vektlegge dette i undervisning. Dette gjør at kunnskap om programmering i skolen er kommet på den norske forskningsagendaen. Formålet med denne studien er å belyse hvilke kjennetegn som finnes på misoppfatninger elever har i arbeid med programmering, og av den grunn skal jeg undersøke følgende problemstilling: «Hva kjennetegner 10. klasse elevers misoppfatninger i arbeid med matematikk i et blokkprogrammeringsmiljø?». For å besvare problemstillingen vil jeg benytte meg av et forskningsspørsmål: «På hvilke områder oppstår misoppfatninger i 10. klasse elevers arbeid med matematikk i et blokkprogrammeringsmiljø?».

Denne studien har et kvalitativt design, og tar utgangspunkt i en case. Deltakerne i casen består av syv elever på 10. trinn, fra én ungdomsskole i Norge. Ved å benytte intervju som en datainnsamlingsmetode, har denne studien avdekket ulike kjennetegn ved elevenes misoppfatninger i arbeidet med programmering. Ved å foreta en tematisk analyse av empirien, er det kommet frem seks områder hvor misoppfatningene finner sted. Disse områdene er sentrale komponenter i begrepet algoritmisk tenkning.

Resultatene fra denne studien viser oss at det som kjennetegner elevenes misoppfatninger ofte stammer fra mangel på gode strategier, høy kognitiv belastning og lite erfaring med programmering. Det er også tegn på at de matematiske kunnskapene elevene innehar har påvirkning på hvorvidt elevene evner å drive problemløsning i programmering. Studiens resultater kan tyde på at det er behov for å arbeide mer, og bedre, med algoritmisk tenkning i skolen.

Nøkkelord: Programmering, algoritmisk tenkning, matematikk, misoppfatninger

Abstract

Programming and computational thinking has become a part of the Norwegian National Curriculum (Utdanningsdirektoratet, 2020), and the subject of mathematics has a special responsibility to address this in teaching. This allows knowledge regarding programming in school to emerge into the research agenda. The purpose of this study is to elucidate what characteristics there is on students´ misconceptions in their work within a programming environment, and for that reason I will investigate the following issue: "What characterizes 10th graders´ misconceptions in their work with mathematics in a block-based programming environment?". To find answer this issue, I will use a research question: "In which areas does 10th graders´ misconceptions occur, in their work with mathematics in a block-based programming environment?".

This study has a qualitative design and is based on a case. The participants in this case consist of seven 10th graders from a middle school in Norway. By using interview as a data collection method, this study has revealed various characteristics of the students´ misconceptions in their work with programming. By doing a thematic analysis of the data collection, six areas of interest have occurred. These areas are key components of the concept of algorithmic thinking.

The results from this study show us that characteristics of students' misconceptions often stems from a lack of good strategies, high cognitive load, and little experience with programming. There are also signs that the mathematical knowledge the students have, has an influence on whether the students are able to solve problems in programming. The results of the study may indicate that there is a need to work more, and better, with algorithmic thinking in school.

Key words: Programming, computational thinking, mathematics, misconceptions

Forord

Tiden fra jeg flyttet til Trondheim for å starte på NTNU like etter videregående, og frem til enden av denne masterutdanningen, har gått utrolig fort. Det heter at tiden flyr i godt selskap, og det har studietiden vært med å bevise. Jeg har vært så heldig å være en del av et klassefellesskap hvor det har vært gøy å ta del i undervisning og forelesning, noe som har holdt opp motivasjonen i perioder hvor studiet har vært krevende. Jeg er utrolig glad for valget av studieretning, og jeg kunne ikke vært heldigere med mine medstudenter. Derfor ønsker jeg å rette en takk til alle som har vært en del av studiehverdagen min de siste fem årene. Sammen har vi kjempet oss gjennom harde perioder, og holdt humør og motivasjon oppe underveis.

Denne masteroppgaven er et resultat av 5 år på institutt for lærerutdanning på NTNU, og for min del markerer den et skifte i livet. Kunnskapen vi har skapt sammen gjennom studieløpet skal nå utøves i praksis, og det er på tide å stå på egne bein. Lærerutdanningen har gjort oss forberedt på dette, og jeg vil derfor rette en takk til instituttet for 5 lærerike år. I tillegg vil jeg takke foreleserne på matematikkseksjonen for spennende og kreative forelesningsmetoder.

Jeg ønsker også å rette en takk til veileder Hermund André Torkildsen, og medveileder Marit Buset Langfeldt, for tilbakemeldinger og innspill i forbindelse med oppgaven. I tillegg ønsker jeg å takke lærerne og inspektøren på den aktuelle skolen, for å la meg komme inn i undervisningen og gjøre forskningsprosjektet. Uten deres tålmodighet og tilpasningsdyktighet under coronaperioden, hadde det blitt vanskelig for meg å gjennomføre prosjektet. Ikke minst ønsker jeg å rette en stor takk til deltakerne i studien, som frivillig har tatt del av prosjektet.

Til slutt ønsker jeg å takke familie og venner for støtten gjennom arbeidet med masteroppgaven. I det som kan føles som en ensom og rigid prosess, har de gode og oppløftende ordene vært motiverende for meg.

Trondheim, mai 2022

Siv Andrea Gulaker

Innhold

Figurer

Tabeller

1 Innledning

Denne masteroppgaven setter søkelys på programmering i skolen. I studien undersøkes det hvilke misoppfatninger som kan oppstå når elever arbeider med matematiske oppgaver i et blokkbasert programmeringsspråk. Formålet med studiet er å kartlegge hva som kjennetegner misoppfatninger som oppstår hos elever på 10.trinn, og på hvilke områder de ulike misoppfatningene finner sted. Hensikten med dette er å skape et bilde av hvilke faktorer som påvirker elevenes læring når de arbeider med programmering, samt øke en bevissthet rundt dette hos lærere som skal undervise elever i programmering.

1.1 Problemområde

Samtidig som verden til stadighet blir mer digitalisert og automatisert, blir også elevenes verden i større grad påvirket av teknologi og ulike fremstillinger av teknologi. Elevens hverdag preges av teknologiske produkter, og fritidsaktiviteter har i større grad fått en teknologisk vinkling. Elevene snakker om e-sport i friminuttene, og skolehverdagen preges av nettbrett eller digitale hjelpemidler som alle elever skal kunne ta i bruk. Fremtidens arbeidsplasser vil høyst sannsynlig preges av algoritmisk tenkning og teknologi, og av den grunn er denne tematikken viktig å få inn i skolen (Barr & Stephenson, 2014). Etter flere tiår med en ekstrem utvikling av den digitale verden, har programmering blitt iverksatt i den nye læreplanen fra 2020 (Utdanningsdirektoratet, 2020). Fagfornyelsen nevner eksplisitt programmering som en del av skolens undervisning. Etter 10. trinn er det et mål at elevene skal kunne:

«Utforske matematiske egenskaper og sammenhenger ved å bruke programmering» (Utdanningsdirektoratet, 2020)

I tillegg ser man i deler av fagfornyelsen at programmering implisitt nevnes, ved bruk av begreper og metoder som er knyttet til programmering og algoritmisk tenkning. Eksempler hvor programmering implisitt nevnes i kjerneelementene i læreplanen for matematikk;

«Algoritmisk tenkning er viktig i prosessen med å utvikle strategier og fremgangsmåter for å løse et problem og innebærer å bryte ned et problem i delproblem som kan løses systematisk» (Utdanningsdirektoratet, 2020),

eller:

«Abstraksjon i matematikk innebærer at elevene gradvis utvikler en formalisering av tanker, strategier og matematisk språk» (Utdanningsdirektoratet, 2020)

Med dette følger det også en økende kompetanseheving for lærere, og kunnskaper om digitale verktøy, og spesifikt programmering, er nødvendig for å kunne utøve den nye læreplanen. Sammenlignet med andre matematikkdidaktiske felt, er det ikke funnet like mye norsk forskning på programmering i skolen, men Gjøvik og Torkildsen skrev i 2019 en artikkel som tok for seg begrepet «algoritmisk tenkning». Slik de beskriver i sin artikkel, kan det bli problematisk når engelske begrep skal innføres i norsk matematikkdidaktikkforskning. Derfor er det viktig at programmeringsfeltet er under utvikling også i Norge, og at man gjennom gjentatte studier kan utvikle et innholdsrikt fagfelt med ny og variert kunnskap. Av den grunn ønsker jeg å bidra til forskningsfeltet ved å rette min studie mot programmering i matematikkdidaktikk.

1.2 Tidligere forskning

Allerede på 80-tallet ble computational thinking (norsk: algoritmisk tenkning) introdusert som et begrep, gjennom boken MINDSTORMS av Seymour Papert (1980). Papert la vekt på to områder knyttet til computational thinking: Hvordan algoritmisk tenkning kan benyttes for å skape ny kunnskap, og hvordan man kan bruke datamaskiner for å forbedre tenkemåter og endre tilgangen til kunnskap. I etterkant av Papert´s (1980) introduksjon av begrepet algoritmisk tenkning, ble det gjort undersøkelser hvor man satte søkelys på hvilke konseptuelle utfordringer elever kunne ha i arbeid med programmeringsspråket LOGO og BASIC (Grover & Basu, 2017). Likevel ser man i etterkant at det er få studier som undersøker elevenes misoppfatninger i introduksjonsdelen av algoritmiske konsepter. Studien Grover og Basu (2017) har gjennomført indikerer at elever på mellomtrinn og ungdomsskole har misoppfatninger knyttet til elementære aspekter i programmering, og med disse resultatene fremhever de et behov for en profesjonell utvikling av læreres konseptuelle forståelse av disse grunnleggende elementene.

I likhet med andre felt innenfor matematikkdidaktikk, er forskere innenfor CS (computational science) interessert i å finne ut av hvordan en kan støtte opp elevenes læring i algoritmisk tenkning (Grover et al., 2017). Problemet for forskerne er at det ikke er like tydelig hvordan man skal oppdage elevenes forståelse og oppfattelse av programmering, som igjen fører til at elevenes læring blir understimulert og mindre forsket på (Grover et al., 2017). De peker derfor på et behov for å undersøke hvordan en skal måle og støtte opp om utviklingen av de ferdighetene algoritmisk tenkning legger til grunn. Dette blir også nevnt i artikkelen til Grover & Basu (2017), hvor det kommer frem at det er nødvendig med pedagogiske strategier og vurderinger som er utformet for å kunne måle elevens forståelse og misoppfatninger knyttet til programmering. Med andre ord kan man se at det er et behov for slik forskning i skolen, og denne kunnskapen vil derfor kunne føre til store implikasjoner på pedagogikken i klasserommet.

I læreplanen slik vi kjenner den i dag, er grunnleggende ferdigheter en viktig del av læringen. Disse ferdighetene skal være med å gi elevene grunnleggende kunnskaper i deler av hverdagen som anses som elementære for å lære, forstå og benytte kunnskap. I tillegg beskrives disse ferdighetene som betydningsfulle for at elevene skal kunne ta del av arbeid og samfunnsliv, og for å kunne delta i utdanning (Kunnskapsdepartementet, 2017, s. 12). Tabesh (2017) har skrevet en artikkel med navn «Computational Thinking: 21st century skill», som poengter hvordan algoritmisk tenkning er blitt en ny ferdighet elever må lære seg å benytte, for å kunne møte det teknologiske samfunnet. På likt grunnlag som grunnleggende ferdigheter, kan også algoritmisk tenkning anses som en ferdighet skolen behøver å lære elevene. Mye tyder på at teknologien og kunnskapen vi har om teknologi vil utvikle seg i årene fremover, og for at elevene skal kunne ta del i samfunnsliv og jobb, vil det være relevant å ha ferdigheter innenfor algoritmisk tenkning. Dette er også noe Wing (2006) påpeker i sin artikkel. Av den grunn er det viktig at skoler og lærerutdanningene setter søkelys på nettopp dette feltet, og de mulighetene algoritmisk tenkning kan skape i skolesammenheng.

1.3 Formulering av problemstilling

Det er tydelig at behovet for læreres kompetanse innenfor programmering i skolen er økende, noe som både implisitt og eksplisitt kommer frem i fagfornyelsen. Tidligere litteratur anmoder et ønske om mer forskning på dette temaet, til tross for at tematikken i seg selv ikke er «ny». Som lærer vil det stor grad være behov for å ha kompetanse i programmering og algoritmisk tenkning, men det vil også være nødvendig å vite hvordan

elevene jobber med dette, og hva som kan være ulike fallgruver underveis. For å rette søkelys på dette, har jeg i min studie valgt å betrakte elevenes misoppfatninger, hvor problemstillingen lyder slik:

«Hva kjennetegner 10. klasse elevers misoppfatninger i arbeid med matematikk i et blokkprogrammeringsmiljø?»

For å kunne belyse problemstillingen på en best mulig måte, har jeg valgt å ta utgangspunkt i et forskningsspørsmål:

> FS: På hvilke områder oppstår misoppfatninger i 10. klasse elevers arbeid med matematikk i et blokkprogrammeringsmiljø?

Forskningsspørsmålet ses på som relevant for å best mulig kunne skape et bilde av hvilke faktorer som kjennetegner elevenes misoppfatninger, da formuleringen av forskningsspørsmålet skaper en mer konkret tilnærming til en større og mer abstrakt problemstilling. I både problemstilling og forskningsspørsmål, blir begrepet «10. klasse elever» benyttet, noe som avgrenser oppgavens område. Begrepet «misoppfatninger» skaper rom for å gå i dybden på individers oppfatning av en tematikk, og av den grunn er dette en kvalitativ studie (Bryman, 2016).

1.4 Studiens oppbygging

Denne studien består av seks kapitler, hvor kapittel én er innledningskapittelet. I innledningskapittelet har jeg presisert problemstillingen, samt belyst temaområdet for studien ved bruk av tidligere forskning og relevante refleksjoner. I kapittel 2.0 vil jeg gjøre rede for litteratur som anses som relevant for studien, og gjøre leseren bekjent med begreper som vil være fordelaktig med hensyn på forskningsgrunnlaget videre i studien. I kapittel 3.0 vil jeg begrunne de metodiske valgene som foreligger for denne studien, og redegjøre for studiens analysemodell. Det vil også bli vurdert ulike betraktninger i form av etiske hensyn og kvalitetsspørsmål. Videre vil det i kapittel 4.0 bli gjort en analyse av de funnene som er relevante for å besvare problemstillingen, hvorpå resultatene vil bli diskutert i lys av relevant litteratur i kapittel 5.0. I tillegg vil det også drøftes mulige årsaker til resultatene i studien. Avslutningsvis vil jeg oppsummere resultatene i studien, og i kapittel 6.0 vil studiens begrensninger og muligheter for videre arbeid bli tatt opp.

2 Teoretisk bakgrunn

I denne studien søker jeg å finne ut av hva som kjennetegner 10.klasse elevers misoppfatninger i arbeid med matematikk i et blokkprogrammeringsmiljø. For å gjøre leseren kjent med bakgrunnen for min vurdering av problemstillingen, vil jeg i dette kapitlet ta for meg begreper, definisjoner og formuleringer knyttet til tematikken for oppgaven. I den forbindelse må jeg redegjøre for hva misoppfatninger er, og hvilke misoppfatninger en kan finne i elevers arbeid med programmering. Det vil også bli sett på begreper som har som hensikt å belyse områder funnene kan organiseres i, og som videre benyttes for å analysere det empiriske materialet. Jeg vil begynne med å kort berette hva som inngår i programmeringsbegrepet, i tillegg til å presentere programmeringsspråket elevene benyttet seg av i arbeid med problemet. Deretter vil jeg gjøre rede for hvilken definisjon som benyttes av begrepene algoritmisk tenkning og computational thinking i oppgaven, hvor jeg avslutningsvis illustrerer dette i en tabell. Videre går jeg inn på hvordan en kan organisere kunnskaper elevene har i programmering, og til slutt vil jeg komme med en redegjørelse på hva som kan føre til elevers misoppfatninger i arbeid med programmering.

2.1 Hva er programmering?

Den første programmerbare enheten stammer så langt tilbake som fra 1822, og ble skissert av Charles Babbage (Haraldsrud, 2019). Babbage´s arbeid dannet et konsept som ble videreutviklet av blant annet Ada Lovelace, som regnes som den første programmereren (Haraldsrud, 2019). Omtrent hundre år etterpå beskrev Alan Turing datamaskiner som tok utgangspunkt i 0-ere og 1-ere (det binære tallsystemet), og ikke lang tid etterpå ble den første elektromekaniske datamaskinen produsert (Haraldsrud, 2019). På under hundre år har vi gått fra å benytte det binære tallsystemet til mer komplekse høynivå-språk, og det er når en utarbeider et program med et språk som en maskin forstår, at man bedriver programmering (Bueie, 2019). Sevik (2016) beskriver programmering som det å lage instruksjoner til en datamaskin, og inkluderer både å skrive koden, men også å designe løsninger. Sevik (2016) viser også til at begrepet «koding» ofte blir brukt i stedet for programmering, og er kanskje mer typisk for skolen. I denne studien vil jeg derimot benytte begrepet programmering, da det begrepet er gjengående i litteraturen.

2.1.1 Scratch

Da programmeringsspråket Scratch ble opprettet var formålet å skape en tilnærming til programmering som kunne appellere til alle, uavhengig av bakgrunn, etnisitet eller kunnskaper (Resnick et al., 2009). Scratch ble lansert i mai 2007, og fungerer som en nettbasert side hvor man kan dele, endre og videreutvikle programmeringsprosjekt. Den primære mottakergruppen er barn i alderen 8 til 16 år (Resnick et al., 2009), hvor hensikten er at barna skal lære matematisk og algoritmisk tenkning, samtidig som de får utfoldet sin kreative og systematiske tankegang i et fellesskap.

Inspirasjonen til hvordan Scratch er bygd opp kommer av arbeid utført i tilknytning til Lego Company, og utviklingen av Lego Mindstorms (Resnick et al, 2009). Om man gir barn en pose med legoklosser, vil de automatisk ta ut deler av klossene og begynne å bygge. Etter

hvert som barna bygger, vil nye ideer oppstå, og de vil utvikle ideene sine. Denne tankegangen er grunnlaget for hvordan Scratch er bygd opp (Resnick et al, 2009). Språket Scratch baserer seg på er dannet av grafiske blokker som kan settes sammen for å skape ny mening, og akkurat som med legoblokker er utformingen av de ulike blokkene skapt slik at man kan finne forslag til hva man kan knytte de sammen med (Resnick et al, 2009). På denne måten vil man kunne lage et program som gir syntaktisk mening, uten å nødvendigvis ha forståelse for syntaks i programmeringsspråk.

2.1.2 Funksjoner i programmering

Jeg vil beskrive noen av funksjonene som er blitt brukt i arbeidet med programmeringsspråket i denne studien. I programmering er en *variabel* en størrelse som kan endres (Haraldsrud, 2019). I Scratch vil en slik variabel ha som hensikt å lagre, motta og oppdatere verdier (Brennan & Resnick, 2012). Om man skal bruke formelen 2x + 4, kan man i Scratch legge inn en variabel som heter «x», og endre verdien på denne underveis. *Lister* har samme hensikt, men i motsetning til variabler er dette en funksjon som gjør at man kan opprettholde en mengde av variabler på en oversiktlig måte (Brennan & Resnick, 2012). *Operatorer* er funksjoner som gjør at man kan utføre matematiske manipuleringer, slik som å addere eller dividere dataene. Blokkene er utformet ved at man kan velge benytte de ulike regneartene, i tillegg funksjoner som <,>, og =. Betingelser, eller *vilkår*, gjør at en kan ta beslutninger basert på spesifikke forhold. Typisk for disse er såkalte «hvis-tester», som gjør at en bestemmer noe så lenge vilkårene man har satt er oppfylt (Brennan & Resnick, 2012). Begrepet «hvis-test» blir brukt på norsk av Haraldsrud (2019). En siste elementær funksjon i Scratch, er *løkker*. Løkker gjør at en kan foreta en sekvens opptil flere ganger (Brenn & Resnick, 2012). Hvis man skal lage en kode i Scratch som gjør at man skal utføre en bevegelse 10 ganger, kan man velge å legge denne koden inn i en løkke, is stedet for å gjenta koden 10 ganger.

2.2 Definisjoner av algoritmisk tenkning

Begrepet algoritmisk tenkning defineres ulikt basert på ulik litteratur (Tedre & Denning, 2016 i Gjøvik & Torkildsen, 2019). I Fagfornyelsens kjerneelementer i matematikk, beskrives algoritmisk tenkning som et sentralt element i det å utvikle strategier og fremgangsmåter i problemløsning (Utdanningsdirektoratet, 2020). I annen litteratur, slik som hos Brennan & Resnick (2012), defineres begrepet ved hjelp av tre ulike nøkkelkategorier: computational concepts, computational practices og computational perspectives. Gjøvik & Torkildsen (2019) på sin side, definerer algoritmisk tenkning ved bruk av fem mindre begrep. Med andre ord er det flere måter å beskrive algoritmisk tenkning på, og som Barr og Stephenson (2011) poengterer, finnes det ikke en allment akseptert definisjon for begrepet per nå. For å klargjøre hvilken definisjon dette studiet baserer seg på, velger jeg videre i dette kapitlet å fokusere på definisjoner jeg kommer til å benytte meg av i oppgaven.

2.2.1 Algoritmisk tenkning (AT)

Ifølge Gjøvik & Torkildsen (2019) kan det være vanskelig å sammenligne den norske bruken av begrepet «algoritmisk tenkning», med det engelske begrepet «computational thinking» (CT). Mye av grunnen til dette er fordi det engelske begrepet «algoritmic thinking» er en komponent i beskrivelsen av computational thinking, samtidig som den direkte oversettelsen «algoritmisk tenkning» i norsk kontekst ses på som ekvivalent med computational thinking (Gjøvik & Torkildsen, 2019). Mangelen på et godt norsk begrep

som erstatning for computational thinking, kan derfor skape misforståelser når begrepene blir brukt om hverandre.

Likevel skriver Gjøvik og Torkildsen (2019) at en norsk definisjon ikke er noe annerledes enn den engelske, men påpeker at en avklaring rundt begrepene «algoritmic thinking» og «algoritmisk tenkning» er nødvendig. Basert på dette beskrives algoritmisk tenkning (AT) som et begrep sammensatt av fem ulike elementer (Gjøvik & Torkildsen, 2019): *Abstraksjon*: som omhandler å kunne abstrahere essensen av ulike tilfeller, og gå bort fra hva som er irrelevant. *Algoritmebehandling*: som går ut på å følge og forklare instruksjoner trinnvis. *Generalisering*: å kunne se mønstre eller sammenhenger for deretter å utlede metoder for dette som fungerer på flere tilsvarende tilfeller. *Automatisering*: det å evne og implementere løsningen inn i et programmeringsspråk, rett og slett ved å automatisere løsningen så menneskers bidrag ikke er like nødvendig. *Dekomponering*: som handler om å kunne bryte opp et problem i mindre deler, og arbeide med hver del hver for seg.

2.2.2 Computatinal thinking (CT)

Når det kommer til det engelske begrepet for algoritmisk tenkning, «Computational Thinking», finnes det flere ulike definisjoner. En av de som har bidratt til å sette søkelys på dette er Jeannette Wing. I hennes artikkel fra 2006 kommer det tydelig frem at computational thinking bør ses på som en fundamental ferdighet alle, uavhengig av studie- eller jobbretning, bør lære seg å benytte. Begrepet i seg selv beskrives som en metode for å løse problemer en ikke er kapabel til å løse på egenhånd (Wing, 2006). Med dette menes det å overføre problemer inn i systemer som gjør jobben enklere for individet som skal utføre det, ved at man tar utgangspunkt i konsepter som er elementære for datavitenskap. Computational thinking omhandler mentale verktøy som gjenspeiler informatikkfeltets vide omfang, og innenfor dette feltet peker hun på begreper som: rekursiv tankemåte, benyttelse av abstraksjon og dekomponering i arbeid med komplekse datamaterialer, separering av hendelser, modifisering av systemer, planlegging av program, feilsøking av kode, med mer (Wing, 2006, ss. 33-34).

Csizmadia et al (2015, s. 5) beskriver CT som evnen til å kunne anvende elementære aspekter innenfor datavitenskap i møte med hverdagslige hendelser, og å benytte disse aspektene til å resonnere og forstå systemer og prosesser innenfor ulike disipliner i natur- og samfunnsvitenskap. For å forstå hva som menes med dette, vil jeg se nærmere på ulike konsepter som ses på som en del av CT. Logisk resonnering ses på som en overordnet ferdighet for å kunne bedrive computational thinking, og gjenspeiles i alle konseptene Csizmadia et al (2015) legger frem i sine beskrivelser. Ved å benytte logisk resonnering vil elevene kunne analysere og dobbeltsjekke fremgangsmåten sin, gjøre antakelser, trekke konklusjoner/skape mening med det de gjør, eller bedrive debugging (feilsøking) av sin egen kode for å rette den opp eller foreslå alternative løsninger (Csizmadia et al., 2015).

I artitkkelen til Csizmadia et al (2015), beskrives CT som et begrep sammensatt av flere viktige komponenter. *Abstrahering* er den første komponenten. Med abstrahering menes det å kunne ta vekk de ubetydelige detaljene for å lage et mer forståelig og oversiktlig system. Det viktigste bak abstraksjon er å vite hvilke detaljer eller elementer som kan tas bort slik at problemet blir enklere å løse (Csizmadia et al, 2015), og det er ferdigheten i å ekskludere de irrelevante detaljene som er essensen bak å mestre abstrahering. En viktig faktor for å lykkes med abstrahering er å finne en hensiktsmessig måte å representere problemet på (Csizmadia et al, 2015, s. 7). Brennan & Resnick (2012) benytter begrepet abstraksjon i samhandling med modulisering, og forklarer dette som å bygge noe stort ved å sette sammen mange, små biter.

Videre beskrives også *evaluering* som en viktig komponent i CT (Csizmadia et al., 2015). Ved å evaluere vil man sikre seg at arbeidet man har gjort er best mulig egnet for problemet man står ovenfor. Denne komponenten understreker viktigheten av å evaluere sine egne løsninger, finne ut om de er passende for sitt formål, og om problemet vil løses best ved bruk av denne løsningen (Csizmadia et al., 2015). Et eksempel på evaluering i arbeid med programmering kan være å vurdere om løsningen en har kommet frem til er enkel for andre å bruke. Hvis man har laget et program som skal gi beskjed når kjøleskapet er tomt, hvordan er det for brukeren av kjøleskapet å benytte seg av dette programmet? Kanskje finner man ut at programmet er vanskelig å sette seg inn i, og revurderer løsningen. I tillegg beskrives også testing som en del av det å evaluere sitt eget arbeid (Csizmadia et al., 2015). Det vil si at en tester koden sin, sjekker om løsningen blir som forventet, og deretter retter opp i faktorer som kan gjøre løsningen bedre. Det samme gjelder også for debugging, hvor en ser etter feil i koden, og gjør endringer for at problemet skal bli løst.

Algoritmisk tenkning ble tidligere beskrevet som en underkategori innenfor det engelske begrepet for algoritmisk tenkning (Gjøvik & Torkildsen, 2019), noe også Csizmadia et al. (2015) velger å benytte seg av i sin beskrivelse av computational thinking. Algoritmisk tenkning ses på som en metode for å komme seg frem til en løsning ved å stegvis benytte seg av algoritmen som trengs, og bør være en naturlig metode for oppgaver hvor samme problem trenger å løses flere ganger. Dette vil si at en ikke trenger å lære seg en ny algoritmisk metode for å løse problemet hver gang det oppstår, men at man kan benytte seg av samme metode flere ganger i lignende situasjoner. Når man først har forstått algoritmen, er det ikke lengre et behov å skulle lære den på nytt når det oppstår nye eller tilsvarende problemer (Csizmadia et al., 2015). I artikkelen til Gjøvik & Torkildsen (2019) kommer det frem at disse trekkene har en likhet med begrepet algoritmebehandling, som er en del av den norske definisjonen av algoritmisk tenkning.

En annen komponent Csizmadia et al. (2015) løfter frem som sentral i algoritmisk tenkning, er *dekomponering*. Dekomponering innenfor programmering og algoritmisk tenkning dreier seg om å tenke på elementer som komponenter av flere deler. Dette betyr at man kan forstå, løse og utvikle delene separate fra hverandre, noe som gjør komplekse problemer lettere å løse (Csizmadia et al., 2015, s. 8). Et godt eksempel Csizmadia et al. benytter i sin artikkel, er det å lage frokost. Å lage frokost kan bli dekomponert inn i individuelle aktiviteter, slik som å koke egg, lage kaffe, eller smøre brødskiven. Disse individuelle aktivitetene kan igjen bli delt inn i nye individuelle aktiviteter, med egne steg underveis. Slik kan man også gjøre når man skal designe et program eller løse et problem ved hjelp av programmering. De dekomponerte delene kan «tas ut» av koden, og endres, evalueres eller utvikles hver for seg, før en eventuelt velger å benytte de koden igjen.

Den siste komponenten Csizmadia et al. (2015) trekker frem for å beskrive algoritmisk tenkning, er *generalisering*. I likhet med matematikk er det også innenfor algoritmisk tekning beskrevet som det å kunne se mønstre og likheter innenfor et fenomen. Generalisering går ut på å kunne løse nye problem basert på tidligere problem, og at man kan bygge på erfaringene man har gjort seg med tilsvarende problem tidligere. Det å kunne forstå hva som skiller det nye problemet med det gamle, for så å anvende likhetene mellom dem, er sentralt for å kunne benytte seg av generalisering (Csizmadia et al., 2015). En

viktig del av å generalisere i programmering er å kunne bruke en algoritme som løser et spesifikt problem, til å løse en hel klasse av tilsvarende problem.

2.2.3 Begrepsavklaring

For å avklare hvilke definisjoner jeg velger å anvende videre i oppgaven, har jeg valgt å sammenfatte de mest sentrale begrepene innenfor algoritmisk tenkning. Disse begrepene er i sin helhet basert på begrepene som er presentert i de foregående kapitlene om algoritmisk tenkning og computational thinking.

Begrepet	**Forklaring av begrep**
Automatisering	Automatisering er en metode for å gjøre det menneskelige bidraget minimalt i programmering. Løsningen blir brakt inn i et programmeringsspråk, og man får større prosedyrer til å bli mindre omfattende for individet ved bruk av datamaskinen. Ved å gjøre dette trenger ikke individene bak systemet å bidra like mye, samtidig som at jobben også blir enklere for den som skal utføre den.
Abstrahering	Abstrahering i programmering innebærer å se bort fra irrelevante opplysninger, men fortsatt ha gode nok opplysninger til å løse problemet. Formålet med abstrahering er å kunne lage noe stort ved å sette sammen flere, små biter. Essensen bak denne komponenten er å være i stand til å ekskludere de detaljene som anses som irrelevante, samtidig som beholder detaljene som er viktige for å løse problemet.
Algoritmebehandling	Algoritmebehandling innebærer at man er i stand til å utføre, og gjerne forklare, en algoritme. For å få til dette må man kunne følge instruksjoner stegvis, og benytte algoritmen på en hensiktsmessig måte. Csizmadia et al. (2015) benytter *algoritmisk tenkning* som et begrep for å forklare dette. Da vises det til at man kan benytte samme algoritme flere ganger for å løse tilsvarende problem, og man slipper å lære en ny algoritme for hver gang.
Evaluering	Evaluering i algoritmisk tekning betyr at man evner å vurdere sitt eget arbeid på en hensiktsmessig måte. Ved å evaluere vil man finne ut om metoden man har benyttet seg av er egnet for problemet, og om det er en fordelaktig fremgangsmåte for å løse problemet. Når man evaluerer sitt eget arbeid stopper man gjerne opp for å stille kritiske spørsmål til det man har gjort, og vurderer om arbeidet er relevant i henhold til formålet. Innenfor evaluering kan vi også trekke frem testing og debugging (feilsøking). Ofte vil man gjennom testing finne ut om løsningen fungerer som den skal. Om den viser seg å ikke gjøre det, kan man debugge programmet for å finne ut hva man kan endre for at programmet skal fungere best mulig.
Dekomponering	Når man dekomponerer noe, bryter man opp et større problem i mindre biter. Å evne å dekomponere innenfor programmering og datavitenskap, betyr at man er i stand til å ta ut de mindre bitene og jobbe med dem separat fra de større delene. Hvis man greier å forstå disse delene etter å ha separert dem, kan man etter å ha utviklet dem benytte de som en del av større

	prosedyrer igjen. Når man etter hvert setter de mindre delene sammen igjen, kan man også løse det opprinnelige problemet.
Generalisering	Generalisering omhandler å kunne se mønstre eller sammenhenger mellom ulike elementer. I programmering vil det å generalisere innebære at man kan løse nye problemer basert på erfaring man har fra tidligere tilsvarende problemer, og benytte likhetene man oppdager på en hensiktsmessig måte. Hvis man benytter seg av en algoritme, kan man benytte denne algoritmen som en generell prosedyre for å løse en hel klasse av tilsvarende problem.

Tabell 1: Begrepsavklaring komponenter i algoritmisk tekning

2.3 Programmeringskunnskap

For å finne ut hva som kjennetegner 10. klasse elevers misoppfatninger i arbeid med matematikk i et blokkprogrammeringsmiljø, er det relevant å beskrive hva som regnes som kunnskaper i programmering. Qian & Lehman (2017, s. 1:3) beskriver læring av programmering som en kjede av kognitive endringer. Det kan være å lære seg nye språklige funksjoner, eller å lære seg konseptene innenfor programmeringsfeltet, for så å lære seg å løse problemer ved bruk av denne kunnskapen (Qian & Lehman, 2017, s. 1:3). Basert på disse endringene, beskriver Qian & Lehman (2017) tre hovedkategorier hvor man kan plassere kunnskap innenfor programmeringsfeltet: syntaktisk kunnskap, konseptuell kunnskap og strategisk kunnskap. Disse tre kategoriene blir benyttet som overordnede kategorier for å beskrive i hvilke områder elevenes misoppfatninger oppstår. I de tre neste underkapitlene vil disse kategoriene beskrives nærmere.

2.3.1 Syntaktisk kunnskap

Syntaktisk kunnskap er kunnskap om hvordan programmeringsspråket er bygd opp, hvilke regler det har og hvilke funksjoner det inneholder (Qian & Lehman, 2017). I Scratch kan et eksempel på syntaktisk kunnskap være å se sammenhengen mellom blokkenes utforming og hvilke blokker som kan settes sammen med en annen blokk. Innenfor tidlig programmering er syntaktisk kunnskap det området hvor det forekommer høyest antall misoppfatninger (Qian & Lehman, 2017), og noen av de vanligste misoppfatningene knyttes opp mot feil bruk av parenteser og andre tegn som er elementære for programmeringsspråk. Slike feil er lettere å oppdage og rette opp i enn andre mulige feil når man programmerer, og anses som mer overfladiske enn de feilene som oppstår innenfor de to andre kategoriene (Qian & Lehman, 2017). Qian & Lehman (2017) peker på at dette kan ha betydning for hvorfor disse feilene lettere blir oppdaget enn andre.

2.3.2 Konseptuell kunnskap

Konseptuell kunnskap beskrives som den forståelsen man har for hvordan programmeringsprinsipper fungerer, og at man er klar over hva som foregår inne i datamaskinen (Quian & Lehman, 2017, s. 1:3). Eksempler på dette kan være at man forstår hvordan en løkke fungerer, eller at man er klar over hvorfor det er nødvendig å starte koden i Scratch med «når ... trykkes». I motsetning til syntaktisk kunnskap, kan misoppfatninger innenfor konseptuell kunnskap føre til større og mer signifikante misoppfatninger (Quian & Lehman, 2017, s. 1:5). Et eksempel på en komponent som kan skape større vanskeligheter hos elevene er variabler. Noen elever kan ha problemer med å forstå at en variabel bare kan ha en verdi på en og samme tid, eller at navnsettingen av variabelen påvirker hvordan elevene benytter den i koden sin. For eksempel kan elevene

tro at verdien på variabelen med navn «stor» er større enn variabelen med navn «liten», til tross for at variablenes navn er vilkårlige og ikke har betydning for verdien variabelen representerer (Quian & Lehman, 2017, s. 1:5).

2.3.3 Strategisk kunnskap

Den siste store kategorien innenfor programmeringskunnskap er *strategisk kunnskap*. Denne kunnskapen omhandler måten man anvender syntaktisk og konseptuell kunnskap på, for å løse et problem i programmering. Den strategiske kunnskapen består av elementer som å debugge, planlegge, og forklare koden (Qian & Lehman, 2017). De påpeker at elevene vil ha betydelige vanskeligheter med å kode eller løse problemer med programmering om de har misoppfatninger innenfor syntaktisk og/eller konseptuell kunnskap, fordi misoppfatninger innenfor strategisk kunnskap har nær tilknytning til disse. Det som er typisk for misoppfatninger innenfor strategisk kunnskap, er at mangelen på gode og veletablerte strategier, samt mangelen på evnen til å se mønstre, fører til vanskeligheter når det kommer til å planlegge, debugge, og skrive koden (Qian & Lehman, 2017, s. 1:6). Det vises også til at nybegynnere ofte har problemer med å forstå oppgaven og dekomponere løsningen slik at den vil passe til problemet de står ovenfor. Ofte kan elever skrive et fullverdig program, men problemene oppstår når elevene møter på uforventede utfordringer i koden. At mange elever i introduksjonsfasen av programmering mangler ferdigheter i å teste og debugge koden sin, er noe som ses på som sentralt (Qiuan & Lehman, 2017).

2.4 Definisjoner på misoppfatninger

I denne studien er formålet å finne ut hva som kjennetegner 10. klasse elevers misoppfatninger knyttet til arbeid i programmering Av den grunn vil det være naturlig å komme med en definisjon på hva som menes med begrepet misoppfatning, og hvilken definisjon denne studien vil ta utgangspunkt i. Qian & Lehman (2017) skiller mellom misoppfatninger i naturvitenskap, og misoppfatninger i datavitenskap. I datavitenskap er det fortsatt ikke utviklet en allmenn akseptert definisjon på hva en misoppfatning er, eller i hvilke områder de oppstår. De viser heller til ulike definisjoner andre artikler har tatt utgangspunkt i. Det som går igjen er ukomplette eller manglende forståelser for programmeringskontekstene, syntaks-feil, misforståelser knyttet til konsepter, utfordringer ved planlegging og debugging av koder, og lignende (Qian & Lehman, 2017). Hovedessensen bak Qian og Lehmans (2017) beskrivelser av misoppfatninger innenfor programmering, er at det finnes flere typer vanskeligheter elevene kan møte på. Et eksempel kan være innenfor løkker, hvor man enten kan ha problemer med å forstå syntaksen knyttet til løkker, eller man kan ha problemer med å forstå hvordan en skal bruke løkker til å løse et problem. I all hovedsak dreier misoppfatningene seg om feil i den konseptuelle forståelsen hos eleven (Qian & Lehman, 2017, s. 1:3).

2.4.1 Misoppfatninger hos elever i programmering

Det finnes artikler som retter søkelys mot hvilke misoppfatninger elever har i møte med programmering i skolen. Grover & Basu (2017) har undersøkt hvilke misoppfatninger elever har i arbeid med blokkbaserte programmeringsspråk, nærmere bestemt Scratch. I denne studien kommer det frem at programmeringsspråk som baserer seg på blokk-system skaper rom for at elevenes syntaktiske misoppfatninger blir færre, men påpeker at den konseptuelle forståelsen fortsatt er manglende. Det kommer frem av studien at blokkbaserte språk, slik som Scratch, kan hjelpe elevene med å forstå de syntaktiske konseptene i programmering bedre, men at det likevel oppstår misoppfatninger når det

kommer til konseptuell og strategisk kunnskap (Grover & Basu, 2017). Grover og Basu (2017) presiserer derfor at det er et behov for å utvikle de pedagogiske strategiene en benytter i skolen, slik at også den konseptuelle og strategiske kunnskapen kan utvikles i positiv retning. I samme studie kommer det også frem at elevene ikke har en dyp og meningsfull forståelse for løkker, eller hva variabler er, og hvordan disse fungerer i en programmeringskontekst. Det blir av den grunn beskrevet som hensiktsmessig å bringe disse aspektene frem i pedagogisk forskning, slik at en kan adressere de faktorene som hindrer elevenes forståelse innenfor datavitenskap og programmering (Grover & Basu, 2017).

Når det kommer til spesifikke områder elever opplever som problematiske, har Cui & Ng (2021) skrevet en artikkel basert på elevenes møte med programmering gjennom matematiske oppgaver. Denne studien tar utgangspunkt i elementer som anses som viktig for algoritmisk tenkning. Resultatene Cui & Ng (2021) legger frem, antyder at misoppfatninger kan oppstå i ulike deler av arbeidet. Blant annet ses det på som sentralt at elevene opplever åpne oppgaver som utfordrende. Videre foreslår de at elevenes tidligere erfaringer med slike oppgaver kan være en mulig årsak til dette. Forfatterne spesifiserer at matematikk på barneskolen ofte omfatter oppgaver som går ut på å «bestemme», «kalkulere» og «løse» problemer (Cui & Ng, 2021), noe som kan hindre elevenes tankegang ved problemløsning. Et annet resultat som kommer frem, er at elevenes strategier for å teste og debugge er lite utviklet, Cui & Ng (2021) ser på den matematiske kunnskapen elevene har fra før som en mulig årsak til dette. Etter som elevene ofte blir instruert i å «sjekke» svarene sine, heller enn å «teste» løsningene sine med andre metoder, kan dette være noe som påvirker den naturlige logikken elevene har når de skal teste koden sin i arbeid med programmering.

Andre faktorer Cui & Ng (2021) belyser i sin artikkel, er misoppfatninger knyttet til programmering i seg selv, samt misoppfatninger knyttet til abstraksjoner. Når det kommer til programmering peker de på at elevenes løsninger ofte bærer preg av matematikk, selv om det i mange tilfeller ikke trenger å være ren matematikk som fører til en god løsning. Et eksempel på dette er at mange elever unngikk, eller glemte, å ha med en funksjon i koden sin som gjorde at resultatet kom opp på skjermen. Selv om elevene hadde gjort oppgaven matematisk riktig, ble den ikke løst riktig i programmeringsspråket (Cui & Ng, 2021). Når det kommer til misoppfatninger knyttet til abstrahering, bruker Cui & Ng (2021) et eksempel hvor elevene hadde vansker med å utlede en formel som gjorde programmet enkelt og oversiktlig. I artikkelen viser de til et eksempel hvor elevene i studien utledet en rekursiv formel, noe som gjorde at de måtte gå gjennom formelen mange ganger for å finne et tall langt bak i tallrekka. Ved å benytte abstraksjon kunne elevene ha forstått hensikten med en eksplisitt formel, og Cui & Ng (2021) antyder at en eksplisitt formel hadde gjort løsningen mye enklere og mindre tidkrevende.

Andre faktorer som kan bidra til å skape misoppfatninger når elevene arbeider med programmering, belyses i artikkelen til Qian & Lehman (2017). Den ene faktoren handler om oppgavens kompleksitet og hvilke krav den stiller til det kognitive arbeidet hos elevene. Et typisk resultat som følge av en oppgave med høye kognitive krav, er at elevene glemmer enkle syntaktiske deler i programmeringen, slik som å benytte en enkel operator eller et tegn i koden. Dette begrunnes av at elevene ved mer komplekse oppgaver, i større grad begynte å glemme viktig informasjon til tross for at de nettopp hadde jobbet med den, noe som førte til økte feil i den syntaktiske delen av arbeidet (Qian & Lehman, 2017). En annen faktor som også nevnes er det naturlige språket. Etter som programmeringsspråk baserer seg på naturlig språk, er det mulig at det oppstår forvirring når elevene skal skrive koder.

Dette er fordi deres opprinnelige oppfatning av språket blir blandet med programmeringsspråket (Qian & Lehman, 2017). Videre peker forfatterne på at nybegynnere i programmering ofte benytter naturlig språk uriktig i arbeidet med programmet. Et eksempel som Qian & Lehman (2017) bringer frem, er at ordet «og» ofte representerer en Boolsk operator i programmeringsspråk, mens det i vanlig språk er en konjunksjon.

Elevenes matematiske kunnskaper er også sentrale for å belyse misoppfatninger elevene har i arbeid med programmering (Qian & Lehman, 2017). Et vanlig eksempel på dette er at elevene forveksler variablenes oppgave i et dataprogram, med oppgaven variabelen har i et algebraisk uttrykk. Notasjonen i programmering kan være til hinder for elevenes forståelse, fordi deres matematiske bakgrunn skaper konflikt med hvordan en benytter seg av ulike matematiske elementer i programmeringskontekst (Qian & Lehman, 2017). En annen faktor som kan skape misoppfatninger hos elevene er at de har unøyaktige eller ukomplette mentale modeller. Det vil si at de har feilaktige forståelser knyttet til den skjulte informasjonen bak prosessen i koden, altså hva som skjer i koden, men som ikke er fysisk observerbart. Slike misoppfatninger ligger dypere, og er ikke like overfladisk som syntaktiske misoppfatninger (Qian & Lehman, 2017).

Videre er også elevenes strategier belyst som et område hvor misoppfatninger kan oppstå (Qian & Lehman, 2017). Til tross for at elevene kan syntaksen til programmet og forstår relevante konsepter knyttet til programmering, kan det likevel oppstå misoppfatninger. Det kan være at elevene har vansker med å resonnere godt nok, eller at programmeringsstrategien som elevene benytter ikke er godt nok utviklet. Et eksempel kan være at eleven ikke evner å forklare eller evaluere sin egen løsning på en god måte, til tross for at eleven har kommet med en løsning (Qiuan & Lehman, 2017). Også faktorer knyttet til miljøet (programmeringsmiljøet eller -språket) kan bidra til å skape misoppfatninger i elevenes arbeid. Det faktum at enkelte programmeringsspråk er såpass ulike, pekes på som en årsak til hvorfor det oppstår forvirring hos elevene. Qian & Lehman (2017) bruker Java som et eksempel, hvor pluss-operatoren «+» kan brukes til mer en enn å bare summere to variabler eller tall. For en som ikke har benyttet seg av Java før, vil denne faktoren være ny og muligens uforståelig.

Den siste faktoren Qian & Lehman (2017) ser på som en mulig påvirker til elevenes misoppfatninger, er lærerens instruksjoner og kunnskap. I noen tilfeller kan læreren også bidra til at det oppstår misoppfatninger hos elevene, og Qian & Lehman trekker frem lærerens strategier for å lære bort, eller bruk av manglende/utilstrekkelig modeller og metaforer som mulige årsaker til dette. I tillegg trekkes mangel på kunnskap hos læreren frem som en faktor elevene kan påvirkes av (Qian & Lehman, 2017). Om læreren selv misoppfatter noe i programmeringen, er det enkelt for å elevene å skape samme misoppfatning.

3 Metode

For å gjennomføre mitt forskningsprosjekt har jeg underveis vært nødt til å ta metodiske valg som har bidratt til å påvirke undersøkelsen min, og resultatet av den. I dette kapittelet vil jeg derfor gå nærmere innpå valgene som er blitt gjort, samtidig som jeg vil begrunne dem. Dette vil jeg gjøre ved å gi en innsikt i hvordan jeg har gått frem for å innhente datamaterialet, hvordan jeg har valgt å samle det inn, og hvordan jeg har gått frem for å analysere det. Jeg vil også reflektere rundt etiske problemstillinger som følge av de metodiske valgene jeg har gjort, og diskutere kvaliteten av studiet i lys begreper som reliabilitet og validitet.

3.1 Kvalitativ tilnærming

Formålet med dette studiet er å finne ut hva som kjennetegner 10. klasse elevers misoppfatninger i arbeid med matematikk i et blokkprogrammeringsmiljø, og har en kvalitativ tilnærming. Målet med er å få en større forståelse rundt deltakernes opplevelse og meningsdannelse knyttet til fenomenet, noe som er et kjennetegn innenfor kvalitativ forskning (Johannessen et al., 2016). Videre beskrives valget om kvalitative studier som hensiktsmessig i forskning hvor man ønsker å forstå fenomenet på en fyldigere og grundigere måte. I følge Bryman (2016) dreier kvalitativ forskning seg om å forstå den sosiale verden ut fra deltakernes oppfatning av den. For å innhente datamateriale som kan gi oss et innblikk i deltakernes opplevelser og meningsdannelser, har man flere ulike innsamlingsmetoder man kan benytte (Johannessen, Tufte & Christoffersen, 2016). I denne studien har jeg derfor valgt å benytte meg av intervju som innsamlingsmetode for å skaffe meg et dypere innblikk i deltakernes tanker og meninger. Det har jeg gjort ved å gjennomføre et undervisningsopplegg, hvor elevene arbeider med oppgaver samtidig som jeg stiller de spørsmål underveis. Ved å benytte disse intervjuene som datamateriale, vil jeg med en induktiv tilnærming forsøke å finne mønstre i deltakernes utsagn, og se om det finnes noen teoretiske begrep som kan knyttes opp mot disse utsagnene.

Kvalitativ forskning henter informasjon om virkeligheten gjennom ord eller språk (Postholm & Jacobsen, 2018, s. 89), og formidler denne virkeligheten gjennom å reprodusere materiale som er blitt sagt, hørt, eller observert. Slik type forskning legger vekt på å forstå fenomener heller enn å forklare de, og har ofte en nærhet til de eller det man forsker på (Tjora, 2021). Typisk for studier med en kvalitativ tilnærming til datamaterialet, er at man benytter seg av en induktiv metode. Dette betyr at man tar utgangspunkt i de empiriske dataene og søker å forstå disse. Med en induktiv tilnærming til dataene ser man etter generelle mønstre, som man senere prøver å beskrive i lys av relevant teori (Johannessen, Tufte & Christoffersen, 2016). Denne studien anses som induktiv fordi jeg vil lete etter mønstre i elevenes utsagn ved arbeidet med programmering og matematikk, og se om jeg kan finne teoretiske begreper som kan være med på å beskrive eller forstå disse mønstrene.

3.1.1 Casestudie

Forskning som benytter seg av casestudie-design kjennetegnes av at forskeren er ute etter å finne unike forekomster innenfor en viss case (Bryman, 2016). I slike studier rettes

oppmerksomheten mot ett eller flere individ, en gruppe, en aktivitet eller lignende (Postholm & Jacobsen, 2018). I denne forskningsoppgaven er utgangspunktet en mindre gruppe elever, som har til felles at de går i samme klasse og er jevnaldrende. Man kan anta at elevene har til dels samme utgangspunkt, og at tilbudet de har hatt gjennom skolegangen har vært tilnærmet likt. Postholm & Jacobsen (2018) legger vekt på at konteksten spiller en sentral rolle i casestudier, og fordi Tjora (2021) beskriver casestudier som studier avgrenset av miljø, vil det av disse grunnene være naturlig å kalle dette en casestudie. I motsetning til nomotetiske vitenskaper, hvor formålet er å generalisere fenomener (Bryman, 2016), kan dette studiet ses på med en idiografisk tilnærming. Med en idiografisk tilnærming til vitenskapen ønsker man å finne unike forekomster av enkeltfenomener (Bryman, 2016). Ved å benytte casestudie som forskningsdesign, vil studien derfor ha en idiografisk tilnærming.

3.2 Datainnsamling

Denne studien tar som nevnt utgangspunkt i kvalitativ forskning, og av den grunn er min innsamlingsmetode basert på metoder som er hensiktsmessig for slik forskning. Målet med datainnsamlingsmetoder i kvalitativ forskning er å innhente ord som kan bidra til å beskrive og forstå menneskers meningsdannelser og handlinger i deres naturlige kontekst (Postholm & Jacobsen, 2018, s. 113). For å kunne innhente denne informasjonen fra elevene som deltok i studien, valgte jeg å intervjue elevene underveis i deres arbeid med matematikk og programmering. Ved å være på den aktuelle skolen, og benytte arbeidsplasser, arbeidsmetoder og arbeidsverktøy som var kjent for elevene, kan man se for seg at elevene var i en naturlig kontekst sett i skoleperspektiv.

3.2.1 Intervju

Innenfor kvalitativ forskning er intervju den mest benyttede metoden for å innhente datamateriale (Thagaard, 2018). Noe som skiller intervju fra andre metoder innenfor kvalitativ forskning, er evnen intervju har til å bringe frem informasjon om intervjuobjektenes følelsesmessige erfaringer og perspektiver knyttet til temaet (Thagaard, 2018). I denne undersøkelsen vil jeg finne ut hva som kjennetegner 10. klasse elevers misoppfatninger i arbeid med matematikk i et blokkprogrammeringsmiljø, noe som fører til et behov for å kartlegge hvordan elevenes tankeprosesser utspiller seg underveis. Derfor vil det være hensiktsmessig å benytte intervju som et verktøy for å fremstille disse tankeprosessene, og komme nærmere innpå elevenes tanker og oppfatninger rundt temaet.

I situasjoner hvor man er i interaksjon med andre mennesker kan man sjelden forutsi utfallet av samtalen. For å gjennomføre intervjuene av elevene valgte jeg derfor å gå for et ustrukturert intervju. I et ustrukturert intervju vil det være et overordnet tema med åpne spørsmål, som kan føre til at det oppstår nye og tilpassede spørsmål underveis som intervjuet foregår (Johannessen et al., 2011). Dette gir rom for at intervjueren kan tilpasse spørsmålene ut fra de ulike situasjonene som kan oppstå under intervjuet. Denne faktoren gjør at jeg som intervjuer kan angripe utsagn fra elevene i de øyeblikkene de oppstår, og intervjuet kan i større grad oppfattes som en uformell samtale. Ustrukturerte intervju kan bidra til å ufarliggjøre samtalen, og skape en mer uformell atmosfære i intervjusituasjonen (Johannessen et al., 2011). Dette anser jeg som en fordel da intervjuobjektene for dette studiet er elever på ungdomsskolen.

3.2.2 Transkribering av datamaterialet

For å kunne benytte innholdet fra intervjuene, lånte jeg to diktafoner fra instituttet. I intervjuene ble disse satt frem slik at lyden tydelig kom frem i opptakene. Begge diktafonene ble brukt for å sikre at lyden ble tatt opp, i tilfelle den ene skulle gå tom for strøm eller ikke fungerte. Lydfilene ble etterpå transkribert, noe som gjorde at jeg fikk en god oversikt over materialet (Kvale & Birkmann, 2009). For å bearbeide materialet, benyttet jeg meg av skriftlige dokumenter i Word, slik at jeg enkelt kunne skrive ned og bla gjennom innholdet i intervjuene. Måten jeg gjorde dette på, var å skrive ordrett det som ble sagt i lydfilene. Denne prosessen var tidkrevende, men samtidig gjorde dette at jeg fikk gjennomgått hele empirien opptil flere ganger. Ulempen med å ta lydopptak for så å skrive ned i etterkant, er at utydelige ord eller setninger kan være vanskeligere å forstå. Etter som jeg ønsket å være så presis som mulig i transkriberingen, er det noen få tilfeller hvor enkelte ord ble tapt i prosessen med å bearbeide materialet.

3.2.3 Rekruttering av deltakere

For å finne en passende gruppe elever for denne forskningsoppgaven var jeg nødt til å kontakte skoleledere i aktuelle skoledistrikt for å forhøre meg om prosjektet var mulig å gjennomføre. Jeg valgte å ta en telefonsamtale til inspektøren på en ungdomsskole jeg hadde sett for meg, og ble deretter satt i kontakt med lærere på det aktuelle trinnet. Mitt ønske for utvalget var i utgangspunktet et tilfeldig utvalg, men etter en samtale med lærerne på trinnet fant vi ut at elevene som ønsket å bidra i undersøkelsen, kunne få mulighet først. Elevene fikk dermed en informasjonsrunde angående prosjektet, hvorpå en «påmelding» skjedde i etterkant. Etter endt informasjonsrunde var det 12 elever som ønsket å være med, og disse fikk alle utlevert et samtykkeskjema som måtte skrives under på av foresatte. Ut fra de 12 elevene som meldte seg på, var det 7 stykker som returnerte samtykkeskjemaet fra sine foresatte. Dette førte til at planen med å ha tre ulike grupper med fire elever i hver ble forkastet, og elevene ble derfor delt opp i en gruppe på 3 og 4 elever. Av disse syv elevene finner vi både gutter og jenter, i tillegg til én elev som har hatt programmering som valgfag.

3.3 Utforming og gjennomføring av programmeringsaktiviteter

For å gjøre leseren oppmerksom på hvordan datamaterialet er blitt dannet, vil jeg i dette kapittelet beskrive oppgavene elevene fikk tildelt, i tillegg til å forklare hvordan gjennomføringen av øktene gikk for seg.

3.3.1 Utforming av oppgaver

I arbeidet med å utforme oppgavene, valgte jeg å gjøre en idèmyldring for å finne ut hvilke typer matematiske oppgaver som kunne passe til prosjektet. Etter som elevene går i 10. klasse var det rom for å dra inn flere tematikker fra læreplanen, da de har, eller i nærmeste fremtid skulle få, kunnskaper om de fleste punktene i læreplanen. Jeg snakket med læreren i forkant av øktene, og h*n syntes ideen om å bruke rente og økonomi som tema var god. Noe av grunnen til dette var også fordi de ikke har holdt på mye med den tematikken tidligere, så elevene ville av den grunn (forhåpentligvis) sitte igjen med et læringsutbytte. Fordi jeg ikke kjente til elevenes matematiske kunnskaper på forhånd, bestemte jeg meg for å prøve ut et opplegg som i større grad ga rom for å utfordre de matematiske kunnskapene. Hvis dette opplegget ikke ga meg den informasjonen jeg trengte, og for mye av tiden ble brukt på selve matematikken (og ikke programmeringen), var planen å justere utformingen av oppgaven til neste gruppe kom.

Jeg valgte å formulere oppgavene i form av steg, heller enn oppgavenummer. Litt av grunnen til dette var at elevene skulle se at man ikke var «ferdig» bare fordi første del av oppgaven var over, men at hele arbeidet var en sammenhengende prosess for å komme frem til en fullstendig løsning. Nedenfor har jeg valgt å legge ved oppgavene gruppene fikk utdelt, hvor hele oppgaven består av fem steg. Det er litt forskjell på oppgavearket til gruppe 1 og 2, noe som gjør at jeg legger ved begge. Den største forskjellen mellom oppgavearkene, er at gruppe 1 ikke fikk spesifikk informasjon om formelen for å regne ut rente. Hos gruppe 1 ble det heller lagt opp til at de skulle finne ut av formelen selv, noe det ble brukt mye mer tid på enn antatt. Derfor gjorde jeg noen små endringer i oppgaven til gruppe 2, slik at vi kunne bruke mer tid på programmering.

Tabell 2: Oppgaveark gruppe 1

Oppgavens kontekst

Du har fått 28 000 kroner i pengepremie i forbindelse med et lotteri, og ønsker å sette inn pengene på en BSU-konto (boligsparing for unge). Hvert år vil summen av det du har inne på bankkontoen øke med renten som banken har satt for BSU-kontoen. I dette tilfellet er renten på 4,0 %.

Steg 1
Skriv ned hvilken informasjon du har bruk for, og sjekk ut hvordan du kan bruke informasjonen til å beregne hvor mye du får i renter etter det første året. Hva er det vi trenger å vite for å beregne dette? *Tips: lag en liste over informasjonen du har, og informasjonen du trenger*

Steg 2
Prøv å beregne hvor mye du vil få i gevinst (summen av renter) det første året, ved bruk av blyant og papir (eventuelt kalkulator hvis du trenger).

Steg 3
Se for deg at du skal lage en kalkulator som kan beregne dette for deg i scratch. Lag en skisse til hvordan denne kalkulatoren kunne ha sett ut. Tegn og/eller skriv ned forslagene du har. (NB: ikke tenk på utseende til kalkulatoren, men funksjonen den skal ha).

Steg 4
Gå på scratch.mit.edu og prøv ut skissen din. Fungerer programmet? Hvis ikke: hvilke endringer kan du gjøre for at det skal fungere?

Steg 5
Det hadde vært interessant å vite hvor mye penger det går an å spare etter et visst antall år, ved å la pengene stå inne på en sparekonto. Lag en skisse til et program som kan beregne hvor mye penger man vil ha på konto etter x antall år, med et innskudd på y kroner og en rente på z %.

- Krav til programmet:

- Brukeren av programmet skal kunne skrive inn et tilfeldig beløp (som skal settes inn på en konto)
- Brukeren av programmet skal kunne skrive inn en tilfeldig rente (etter som denne kan endre seg fra bank til bank)
- Brukeren av programmet skal kunne skrive inn hvor mange år h*n ønsker å la pengene stå inne på kontoen

- *Tips: bruk spørsmål/svar funksjonen, bruk lister og/eller andre funksjoner*

Tabell 3: Oppgaveark gruppe 2

Oppgavens kontekst

Du har fått 28 000 kroner i pengepremie i forbindelse med et lotteri, og ønsker å sette inn pengene på en BSU-konto (boligsparing for unge). Hvert år vil summen av det du har inne på bankkontoen øke med renten som banken har satt for BSU-kontoen. I dette tilfellet er renten på 4,0 %.

*Formelen for å regne ut rente er **innskudd * rente / 100.***

Steg 1
Prøv å beregne hvor mye du vil få i gevinst (summen av renter) det første året, ved bruk av blyant og papir (eventuelt kalkulator hvis du trenger).

Steg 2
Se for deg at du skal lage en kalkulator som kan beregne dette for deg i scratch. Lag en skisse til hvordan denne kalkulatoren kunne ha sett ut. Tegn og/eller skriv ned forslagene du har. (NB: ikke tenk på utseende til kalkulatoren, men funksjonen den skal ha).

Steg 3
Gå på scratch.mit.edu og prøv ut skissen din. Fungerer programmet? Hvis ikke: hvilke endringer kan du gjøre for at det skal fungere?

Steg 4
Det hadde vært interessant å vite hvor mye penger det går an å spare etter et visst antall år, ved å la pengene stå inne på en sparekonto. Lag en skisse til et program som kan beregne hvor mye penger man vil ha på konto etter x antall år, med et innskudd på y kroner og en rente på z %.

- Krav til programmet:
  - Brukeren av programmet skal kunne skrive inn et tilfeldig beløp (som skal settes inn på en konto)
  - Brukeren av programmet skal kunne skrive inn en tilfeldig rente (etter som denne kan endre seg fra bank til bank)
  - Brukeren av programmet skal kunne skrive inn hvor mange år h*n ønsker å la pengene stå inne på kontoen
- *Tips: bruk spørsmål/svar funksjonen, bruk lister og/eller andre funksjoner*

Når øktene skulle begynne, hentet jeg elevene som skulle delta og tok de med til et grupperom som var satt av for anledningen. Her hadde jeg organisert et større bord, slik at alle elevene enkelt kunne diskutere og snakke sammen. I oppstarten fortalte jeg elevene hva som kom til å skje, hva jeg kom til å gjøre underveis, og hva elevene selv skulle gjøre. Jeg ga elevene oppgavearkene slik at de kunne lese over, og ba deretter elevene om å spørre om hjelp eller si ifra hvis noe var uklart. Målet mitt var å ta minst mulig del i elevenes arbeidsprosess, og lot de derfor bruke litt tid på å sette seg inn i, og forstå, oppgaven på egen hånd. Når dette var gjort, begynte prosessen med å skape en uformell samtale med elevene.

3.4 Metode for analyse

For å kunne gjøre en vurdering av hva som kjennetegner 10. klasse elevers misoppfatninger i arbeid med matematikk i et blokkprogrammeringsmiljø, har jeg analysert det empiriske materialet. En stor del av analysen i kvalitativ forskning er selve kodingen, og et vanlig problem innenfor denne forskningen er å redusere datamaterialet til et forståelig omfang empiri (Cohen, Manion & Morrison, 2011, s. 559). Datamaterialet som nå skulle analyseres var på om lag 80 sider, og en omfattende kodingsprosess ble derfor nødvendig. Etter å ha startet prosessen med å markere mindre koder for hånd i transkripsjonene, ble jeg oppmerksom på et kode-verktøy kalt Nvivo. Jeg fant fort ut at å bruke et slikt verktøy ville spare meg for mye tid, samtidig som Nvivos ulike funksjoner også kunne hjelpe meg å se sammenhenger og trekke linjer mellom ulike deler av datamaterialet. Av den grunn forkastet jeg arbeidet jeg hadde begynt med, og startet med blanke ark i Nvivo. For å analysere funnene jeg har gjort meg i prosessen har jeg valgt å benytte meg av tematisk analyse. Dette er fordi denne typen analyse ikke er bundet til et spesifikt teoretisk rammeverk, og kan derfor benyttes til ulike formål (Braun & Clarke, 2006).

3.4.1 Tematisk analyse

Formålet med det Thagaard (2018) beskriver som temaanalyse, er å kunne gå i dybden på enkelte temaer. Ved å gå i dybden vil man utvikle en dypere forståelse av tematikken som er fokusområdet for forskningen, og det kan gjøres ved å sammenligne data på tvers av ulike deltakere (Thagaard, 2018). Når en skal sammenligne data innenfor samme tematikk er det essensielt å forholde seg til koder og klassifiseringer som gir mening på tvers av dataene. Dette kan fungere som et felles utgangspunkt for analyser som søker å forstå fenomener i kontekster eller temaer (Thagaard, 2018), og vil være en relevant fremgangsmetode for mitt arbeid med casestudie.

Da jeg begynte analysen av empirien jeg hadde samlet inn, hadde jeg flere lydopptak som skulle gjennomgås og transkriberes. Selve transkriberingen er en viktig del av det å bli kjent med datamaterialet, og er den første delen av tematisk analyse (Braun & Clarke, 2006). For å transkribere datamaterialet valgte jeg å gå gjennom lydopptakene samtidig som jeg skrev ned alt for hånd i egne dokumenter. Denne prosessen gjorde at jeg fikk finkjemmet hele datamaterialet, og lest hvert eneste ord opptil flere ganger. Ved å bli kjent med datamaterialet på denne måten, fikk jeg underveis anledning til å skrive notater med ideer om hva slags koder som kunne være relevante senere. Disse ideene benyttet jeg meg av når jeg skulle inn i neste fase av analysen, som er å opprette innledende koder for datamaterialet (Braun & Clarke, 2006). Ved å opprette disse kodene, fikk jeg ved hjelp av Nvivo-verktøyet raskt en bedre oversikt over hvilke koder som var gjentakende for materialet, og hvilke koder som ikke forekom like ofte. Denne delen av prosessen er en viktig del av analysen, etter som jeg får organisert datamaterialet mitt i meningsfulle

grupper (Braun & Clarke, 2006). Etter å ha gått gjennom alt av datamateriale var dette de innledende kodene jeg satt igjen med til slutt:

Algoritmebehandling	Automatisering	Forståelse programmering	Misoppfatning vilkår	Overgeneralisering
Dekomponering	Forståelse lister	Forståelse variabler	Misoppfatning variabel	Planlegging
Evaluering	Forståelse løkker	Misoppfatning	Misoppfatning syntaks	Debugging
Generalisering	Forståelse syntaks	Misoppfatning lister	Misoppfatning matematikk	Testing
Abstrahering	Forståelse matematikk	Misoppfatning Løkker	Misoppfatning programmering	Programmering teknisk
Lærerforklaring				

Tabell 4: Innledende koder fra datamaterialet

I denne tabellen ser vi at begrepene innenfor algoritmisk tekning står som egne kategorier. Disse kategoriene valgte jeg å ta med slik at jeg enkelt kunne legge inn utsagn som åpenbart passet inn hos én av dem. Ellers har jeg valgt å bruke elementer hvor det er mulig at elevene danner seg misoppfatninger, slik som i løkker eller variabler. Jeg har også valgt å ta med de utsagnene hvor elevene viser at de forstår aspektene knyttet til disse elementene.

Ofte når man koder et datamateriale, kan det hende at enkelte utsagn eller koder oppstår innenfor flere koder (Cohen et al., 2011). Et eksempel på et utsagn som jeg har valgt å plassere innenfor to ulike koder er dette:

> Elev 1: kan du ikke bare sette inn tall ... Er det ikke en sånn «sett inn» ...

Dette utsagnet valgte jeg å plassere innenfor koden «forståelse variabel», fordi eleven ser at det er hensiktsmessig å bruke en variabel for å få ønsket resultat. I tillegg plasserte jeg utsagnet i koden «forståelse syntaks», fordi eleven forstår at programmet trenger en egen funksjon som heter «sett inn» for å kunne benytte variabelen i programmet.

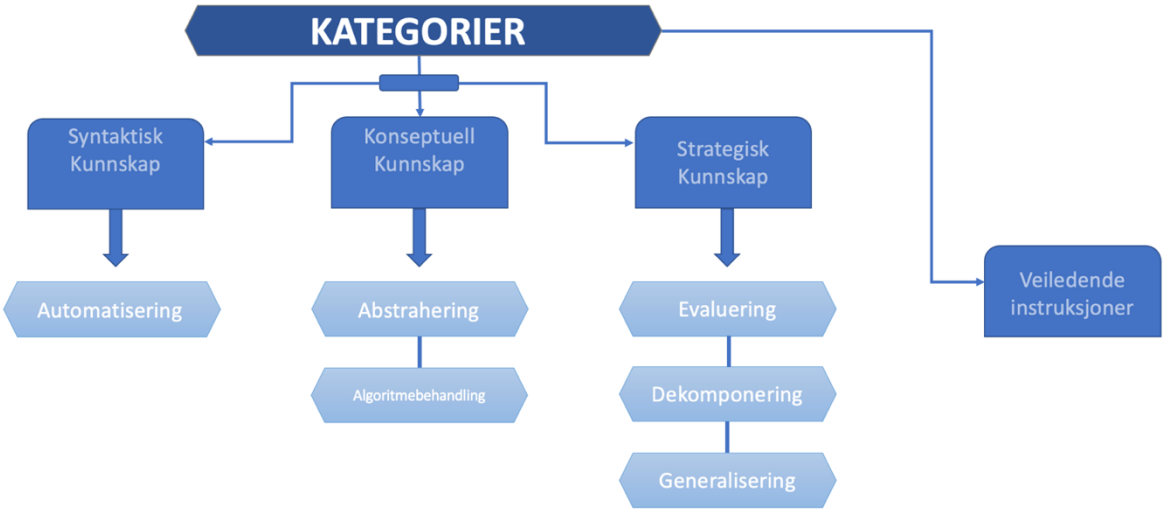
Etter å ha plassert alt av relevante utsagn i de innledende kodene vist i tabellen ovenfor, ble neste oppgave å kategorisere disse kodene i større temaer (Braun & Clarke, 2006). Når jeg så over de ulike kodene jeg hadde laget, fant jeg ut at en metode å kategorisere disse kodene på kunne være å ta utgangspunkt i begrepene algoritmisk tenkning og computational thinking. Disse begrepene baserer seg på ulike komponenter i algoritmisk tenkning og computational thinking, som ble beskrevet i kapittel 2.2.3. De innledende kodene ble derfor fordelt inn i disse seks kategoriene:

Automatisering	Abstrahering	Algoritmebehandling
Evaluering	Dekomponering	Generalisering

Tabell 5: Kategorisering av innledende koder

Etter å ha fordelt alle relevante utsagn inn i kodene basert på komponenter i algoritmisk tenkning og computational thinking, lagde jeg en ny gruppering av koder basert på større

kategoriseringer av elevers misoppfatninger i programmering. Disse kategoriene er syntaktisk-, konseptuell- og strategisk kunnskap. Innenfor disse tre kategoriene har jeg valgt å plassere kategoriene basert på AT og CT. I illustrasjonen nedenfor kommer det frem hvilke kategorier som tilhører hva. Under arbeidet med å kategorisere alle utsagnene, ble jeg oppmerksom på at utsagnene innenfor koden «lærerveiledning» ikke nødvendigvis kunne plasseres innenfor de tre hovedkategoriene. I tillegg var utsagnene innenfor denne koden såpass store og omfattende, at det ble naturlig å beholde dette som en egen kategori uavhengig av de andre. Av den grunn har jeg valgt å benytte fire hovedkategorier, hvor tre er basert på et teoretisk grunnlag, og én er basert på datamaterialet.



Figur 1: Kategorisering av koder fra datamaterialet

I syntaktisk kunnskap finner vi «automatisering», fordi denne komponenten handler om å bringe løsningen på problemet inn i et programmeringsspråk. Ergo er det behov for å ha kunnskaper om språkets oppbygging og funksjon, og å vite hvordan en kan benytte seg av det. I konseptuell kunnskap finner vi «abstrahering» og «algoritmebehandling», da disse komponentene tilsier at en må vite hvordan koden fungerer, og hva som foregår i den. I strategisk kunnskap har jeg plasser «evaluering», «dekomponering» og «generalisering», fordi disse komponentene fordrer at en evner å se mønstre, samt at en har utviklet gode strategier.

3.5 Etiske betrakninger og konfidensialitet

I forskning er samtykke et viktig prinsipp, og det å få en offisiell bekreftelse på at forskningen kan foretas er viktig (Cohen et al., 2011). For å være sikker på at min studie kunne gjennomføres sendte jeg inn en søknad til Norsk Senter for Forskningsdata (NSD), hvor jeg fylte ut et meldeskjema som ble sendt inn for godkjennelse. Dette gjorde jeg fordi studiet skal behandle personopplysninger. Søknaden som ble sendt inn til NSD ble godkjent, og jeg fikk dermed tillatelse til å hente og behandle personversnopplysninger (Vedlegg 2). Cohen et al. (2011) påpeker også at det er viktig å kontakte eventuelle lærere eller kontaktpersoner i skolen om barn skal ta del av forskningen. Dette ble gjort tidlig i forskningsprosjektet, både for godkjenningens del, men også for organiseringens del.

En annen viktig del av forskningen er deltakernes mulighet til å være anonyme, og konfidensialiteten bak informasjon som er blitt samlet inn er derfor svært viktig (Cohen et

al., 2011). Det som kommer frem som desidert viktigst ved anonymitet i forskning, er at informasjonen som er blitt samlet om deltakerne på ingen måte kan avsløre identiteten deres. En deltaker er derfor regnet som anonym så lenge en annen person ikke kan finne ut hvem det er gjennom å lese denne oppgaven. For å ivareta anonymiteten til deltakerne i studien min, har jeg derfor valgt å anonymisere alle navn, både på elever og skole, for å unngå at opplysningene kan knyttes til noen av deltakerne. Et eksempel på et valg jeg gjorde for å sikre anonymiteten, var å unngå å skrive ned navnene som ble nevnt underveis i transkripsjonen. I stedet har jeg valgt å kalle de «Elev 1», «Elev 2» og så videre.

Før jeg begynte innsamlingen av datamateriale fikk deltakerne selv mulighet til å samtykke at lyd- og videoopptak kunne bli benyttet. I tillegg fikk foresatte et samtykkeskjema som de var nødt til å skrive under på, før forskningen begynte. Skjemaet som ble benyttet var tatt direkte fra malen hos Norsk Senter for Forskningsdata (NSD), og er godkjent som samtykkeskjema. Dette skjemaet ligger som vedlegg (vedlegg 1).

3.6 Studiens kvalitet og troverdighet

Selv om forskningens resultat i mange tilfeller kan være svært relevant og nyttig for andre, behøver ikke dette å tilsi at kvaliteten bak forskningen er god. Det finnes ulike måter å vurdere forskningens kvalitet på, annet enn å se på resultatet (Postholm & Jacobsen, 2018). For at forskningen skal være overbevisende for leseren, vil det være nødvendig å spesifisere hvordan kunnskapen er produsert. Dette er en av de viktigste faktorene for å tydeliggjøre studiens kvalitet (Postholm & Jacobsen, 2018). En måte å gjøre dette på er ved å være transparent, eller gjennomsiktig, i sin forskningsmetode. Det vil si at detaljerte beskrivelser av fremgangsmåte, strategi, og metoder for analyse blir beskrevet, slik at en utenforstående kan vurdere forskningens troverdighet (Thagaard, 2018). I dette kapitlet tar jeg derfor for meg relevante begreper knyttet til forskningskvalitet, og sikter på å skape en så transparent studie som mulig.

3.6.1 Studiens validitet

Validitet, som ofte blir omtalt som gyldighet (Postholm & Jacobsen, 2018), deles inn i to typer: Indre gyldighet og ytre gyldighet. Indre gyldighet dreier seg om hvorvidt det er samsvar mellom virkeligheten vi studerer og begrepene vi benytter for å beskrive den, og i hvilken grad vi har grunnlag for å uttale oss om årsak og virkning ut fra forskningen som ligger til grunn (Postholm & Jacobsen, 2018). For å styrke studiens validitet har jeg derfor brukt mye tid på å sette meg inn i, og forsøke å forstå datamaterialet. Dette vises i form av at jeg opptil flere ganger har gått gjennom transkripsjonene, kodet kategorien flere ganger for å se om det samsvarer med tidligere inntrykk, i tillegg til at jeg har søkt bekreftelse på valgene jeg har tatt av eksterne personer. Denne prosessen har ført til at jeg med høy sikkerhet kan stole på at valgene jeg har tatt for å kategorisere den empiriske dataen, er gjort på en måte som kan bidra til å belyse formålet med denne studien.

Den ytre gyldigheten kjennetegnes av at forskningen er overførbar, altså at resultatene i forskningen på ett vis kan *generalisere* (Postholm & Jacobsen, 2018). Formålet med dette studiet er ikke å generalisere resultatene, da utgangspunktet er en liten gruppe deltakere. Likevel ønsker jeg å gjøre rede for studiens gang, og la leseren få et innblikk i forskningsprosessen. Dette er hva Postholm & Jacobsen (2018) kaller en «naturalistisk generalisering». Dette gjør jeg ved å være transparent i studien i form av at leseren får innblikk i hvordan studien har blitt til, samt hvordan jeg har vurdert mulige problemer som har oppstått underveis. Leseren får også et innblikk i hvordan studien har tatt form, da jeg

gjør rede for hvordan jeg har gått frem for å samle inn data, hvordan jeg har behandlet den, og hvordan jeg har valgt å analysere den. Dette styrker den ytre validiteten til studien.

3.6.2 Studiens reliabilitet

I kvalitativ forskning er det relevant å vurdere forskningens troverdighet, og prosjektets reliabilitet er en av kriteriene for at forskningen er utført på tillitsvekkende måte (Thagaard, 2018, s. 187). Postholm & Jacobsen (2018) beskriver reliabilitet som forskningens pålitelighet, og nevner refleksjon rundt forskerens egne påvirkning, samt synliggjøring av forskningsprosessen, som viktige faktorer for å styrke dette. Ved å ta med leseren gjennom alle prosesser i forskningen, bidrar jeg til å synliggjøre forskningsprosessen. Det skal derfor ikke være tvil om hva som er blitt gjort, og hvilke valg som er tatt underveis i denne studien.

I søken om å skape et gjennomsiktig forskningsprosjekt, ønsker jeg også å reflektere over min posisjon som forsker i innsamlingsprosessen. Hvordan forskeren forstår miljøet som blir studert, kan påvirkes av hvilken relasjon forskeren har til deltakerne i studien (Thagaard, 2018). Av den grunn velger jeg å sette søkelys på hvordan min posisjon som forsker kan ha vært med å påvirke forståelsen av miljøet som er studert. Det er relevant å reflektere over hvordan min rolle kan ha hatt betydning for deltakernes oppfatning av meg, og hvordan relasjonen mellom meg og deltakerne på noen måte kan ha gått på bekostning av forskningens resultater.

Skolen som ble valgt ut for dette studiet er en skole hvor jeg har jobbet som vikar over flere år. Klassen og trinnet som har bidratt i studien, er derimot et trinn hvor jeg har vært til stede svært lite sammenlignet med andre klasser eller trinn. Likevel er det en faktor at flere av elevene kan kjenne meg igjen, både som vikarlærer, men også som en lagspiller gjennom fritidsaktiviteter. Av den grunn har det derfor vært viktig for meg å definere min rolle innad i dette prosjektet før prosjektet ble satt i gang, noe Postholm og Jacobsen (2018) også påpeker som formålstjenlig. Et grep jeg gjorde for å skille mellom rollen som vikar, lagspiller og forsker, var å starte prosjektet med en introduksjon av meg selv og hva prosjektet omhandler. På denne måten fikk elevene se meg i rollen som forsker, og de kunne oppleve en profesjonalitet knyttet til studiet. Formålet med dette var at elevene skulle anse forskningen som seriøs, samt at de skulle oppfatte min rolle som forsker profesjonell.

En viktig del av forskningen er hvordan kunnskapen blir til, og Postholm & Jacobsen (2018) legger vekt på at dette skjer i forbindelse med forskerens møte med konteksten eller forskningsdeltakerne. Kunnskapen som kommer frem i denne studien er basert på møtet mellom meg som forsker, og elever som jeg til en viss grad kjenner fra før. Etter som det er dialogen mellom forskeren og deltakerne som utgjør tolkningen av funnene som blir gjort (Postholm & Jacobsen, 2018), kan derfor dette være med å påvirke resultatet. I denne studien har jeg vært klar over dette gjennom hele løpet, og forsøkt å benytte meg av subjektive tolkninger basert på det teoretiske grunnlaget. Likevel kan man aldri være helt subjektiv i en studie (Postholm & Jacobsen, 2018), og mine personlige tolkninger og forståelser av elevenes utsagn vil alltid være med å legge grunnlaget for resultatene studiet vil gi.

4 Analyse av funn

I denne delen av oppgaven vil jeg ta for meg de funnene som er blitt gjort i henhold til begrepene som beskriver algoritmisk tenkning og programmeringskunnskap. Etter å ha analysert datamaterialet er det gjort funn som kan bidra til å belyse problemstillingen: «*Hva kjennetegner 10. klasse elevers misoppfatninger i arbeid med matematikk i et blokkprogrammeringsmiljø?*». Kapitlet er delt inn i de tre hovedkategoriene beskrevet i kapittel 3.0 Metode. I tillegg har arbeidet og analyseringen av kodene ført til en siste kategori som ikke kan belyses ut fra den øvrige teorien. Denne kategorien har jeg valgt å kalle «Veiledende instruksjoner». Kategorien inneholder alle utsagn hvor læreren har vært nødt til å instruere eller veilede elevene i arbeidet med oppgaven, av årsaker som at elevene har stått fast eller misforstått/manglet elementer for å kunne fortsette arbeidet på egen hånd.

4.1 Syntaktisk kunnskap

Dataene som er innhentet i forbindelse med denne studien peker på at elevene har ulike misoppfatninger i ulike faser av arbeidet med programmering. Under kategorien «syntaktisk kunnskap» har jeg valgt å plassere utsagn som kjennetegnes av at kunnskapen til elevene tyder på misoppfatninger rundt å forstå språket (Scratch), hvordan det er oppbygd og hvilke regler det har. Misoppfatningene som blir belyst gjennom utsagnene kan plasseres innenfor kategorien «automatisering», en av komponentene i algoritmisk tenkning.

4.1.1 Automatisering

I denne kategorien vil jeg legge frem de funnene som kan kjennetegnes av at elevene møter på misoppfatninger når de skal automatisere koden sin, altså når de skal bringe løsningen på oppgaven inn i programmeringsspråket. Utsagnene som blir fremhevet baseres på problemer elevene har hatt med å bringe løsningene sine inn i Scratch, og knyttes i all hovedsak opp mot usikkerhet og misoppfatninger når det gjelder syntaks.

> **Utsagn 1**
>
> *Lærer: Skal vi se ... Så du har tatt frem ei løkke*
>
> *Elev 1: Men jeg vet bare ikke helt hvordan jeg skal sette det sammen (Gruppe 2)*
>
> **Utsagn 2**
>
> *Elev 3: Den skal være svar på der, den skal være svar på der, og den skal være svar på der. Men jeg vet ikke hvordan man kobler det opp, skulle jeg til å si ... (Gruppe 2)*

Både i utsagn 1 og 2 ser vi at elevene sliter med å finne ut hvordan koden deres i Scratch skal henge sammen. Den ene eleven forklarer at h*n ikke vet hvordan koden skal «settes sammen», og den andre eleven vet ikke hvordan h*n skal «koble det opp». Arbeidet til elevene stoppes i disse situasjonene av mangel på å vite hvordan syntaksen i programmeringsspråket er bygd opp, noe som hindrer elevene i å gå videre i problemløsningen.

Utsagn 3

Elev 2: Så vi skal ikke bruke en sånn her? (peker på skjerm). En sånn «hvis/ellers»?

Elev 1: Det kommer jo an på hva ... (Gruppe 1)

I dette tilfellet (utsagn 3) ser det ut til at eleven virker å ikke vite hva slags funksjon vilkår har i programmering, og misforstår hvordan den kan brukes for å løse problemet. At eleven har problemer med å forstå språket det programmeres i, får i så tilfelle konsekvenser for evnen eleven har til å automatisere koden.

Utsagn 4

Elev 1: Ja hvor finner du den da?

Elev 2: Den er på hend... Styring ... Ser jeg for meg

Elev 1: Okei

Elev 2: Eller hendelser

Elev 1: Ja, eller hendelser ja

Elev 2: Det var fargen jeg så på ... (Gruppe 1)

Utsagn 5

Lærer: Du bruker de samme klossene, men du må endre variabelen

Elev 1: Okei, da må jeg ... Variabelen, er det det som er her?

Lærer: Variabelen er der (peker på fargen til variabelen). (Gruppe 1)

Utsagn 6

Elev 2: Ja det er noe her ... «Set y til»

Lærer: Hvilken type blokk er det der da? ... (ingen respons) ... Det var «bevegelse»

Elev 2: Vet ikke helt hva den egentlig gjør, men ... X og y (Gruppe 2)

I utsagn 4, 5 og 6 ser vi at elevene er usikre på hvor man skal finne riktige funksjoner i programmeringsspråket. Dette tyder på at elevene ikke er kjent med språket scratch og dets syntaks, noe som kan forhindre elevenes automatisering i arbeid med programmering. Utsagn slik som dette finnes det mange av i datamaterialet, og konsekvensene av denne problematikken har vært at elevene bruker mye tid på å sette seg inn i funksjonene

En gjennomgående faktor i elevenes arbeid med oppgaven, og en betydelig del av datamaterialet, tyder på at elevenes usikkerhet rundt programmeringsspråket har bidratt til å skape misoppfatninger eller stopp i arbeidet med å løse problemet. I flere deler av datamaterialet finner vi utsagn hvor elevene antyder eller spesifiserer at de er usikre på hva slags type blokker eller komponenter de skal benytte seg av. I tillegg ser man også at det gjentatte ganger er problematisk for elevene å vite hvordan blokkene skal settes sammen for å få ønsket resultat, noe som igjen fører til forvirring og usikkerhet videre i arbeidet.

4.2 Konseptuell kunnskap

I hovedkategorien «konseptuell kunnskap» vil det komme frem utsagn som kjennetegnes av elevenes mangler på å forstå elementære programmeringsprinsipper, og deres mangler på å forstå hva som skjer inne i de ulike elementene i koden. I denne kategorien har jeg valgt å plassere to komponenter fra algoritmisk tenkning. Dette er «abstrahering» og «algoritmebehandling». I underkategorien «abstrahering» er det ikke gjort like mange funn av misoppfatninger som i «algoritmebehandling», men likevel tror jeg dette kan bidra til å få et tydeligere svar på problemstillingen.

4.2.1 Abstrahering

I «abstrahering» ses det på utsagn hvor det er blitt feilvurdert eller misoppfattet hvilke opplysninger som er relevante og ikke for oppgaven. I denne kategorien er det færre funn som spesifikt kan knyttes til abstrahering, om vi sammenligner med andre underkategorier.

Utsagn 7

Elev 3: Må vi ikke gjøre det der flere ganger? Slik at vi setter den der (peker på skjerm), så kommer det en ny sånn spørsmål, og så kommer den der på nytt igjen (peker på skjerm).

Elev 1: Hvorfor skal den det?

Elev 3: Nei jeg vet ikke, hehe

Lærer: Nei, men det er godt spørsmål det! Trenger vi å ha den to ganger i koden?

Elev 2: Vet ikke

Lærer: Trenger kalkulatoren å regne ut dette flere ganger? (Gruppe 1)

I utsagn 7 tenker elev 3 at formelen de har benyttet må gjentas flere ganger i løpet av koden, fordi de har ulike spørsmål som skal gi de ulike svar. Det eleven ikke ser er at det som er inne i formelen kan endres ved å endre variablene. Med andre ord kan koden bli kortere hvis eleven unngår å gjenta enkelte prosesser, og ikke tar med formelen. Eleven ser i dette tilfellet ikke essensen bak koden, og sliter med å ekskludere unyttige detaljer.

Fraværet av funn i denne kategorien kan skyldes oppgavens ordlyd, som har ført til at abstrahering ikke har vært nødvendig for å komme frem til en hensiktsmessig løsning. En annen årsak kan også være at elevene ikke har benyttet seg av abstrahering, og at det av den grunn ikke kommer frem funn som kan være relevant for denne kategorien. Om det siste tilfellet gjelder kan det tyde på at elevene mangler forståelse for prinsippet bak å abstrahere, og er i så fall et kjennetegn på en misoppfatning.

4.2.2 Algoritmebehandling

De funnene som vises til i kategorien «algoritmebehandling» er utsagn som kan tyde på misoppfatninger rundt det å forstå, anvende eller endre algoritmer. Her vil det komme frem utsagn hvor elevene ikke ser meningen bak algoritmen, eller ikke forstår hvordan den kan benyttes andre steder i programmet.

Utsagn 8

Elev 1: Jeg sliter litt med å få til å dele ... Gevinst pluss innskudd delt på hundre ... Eller, jeg får det ikke inn i samme sånn ...

Lærer: Sånn? Men gevinst pluss ... Hvorfor skal du dele på hundre?

Elev 1: Nei, det var det jeg ikke skulle gjøre (Gruppe 1)

Utsagn 9

Elev 1: Vi må gange med hundre, og så ... Ja, nei! Hvis du skal ha fire prosent da, så må du gange det med hundre og så deler du det på 28 000, er det ikke det da?

Elev 2: Ja, det er firehundre er det ikke? Jo, fire ganger hundre ja

Elev 1: Ja, jeg tror nå hvert fall det er det (Gruppe 1)

I utsagn 8 og 9 er det tydelig at elevene mangler forståelse for formelen de har funnet. I begynnelsen av arbeidet lagde elevene en konkret formel som fungerte på ett spesifikt eksempel. I disse tilfellene har elevene forsøkt å generalisere formelen ved å snu om på den. Dette har gjort at formelen får en helt annen løsning enn hva de i utgangspunktet hadde. Utsagn 8 og 9 kan tyde på at den matematiske forståelsen til elevene påvirker hvordan de arbeider med algoritmen.

Utsagn 10

Elev 1: Jeg får ikke opp gevinsten jeg ...

Elev 2: Men, du får inn 11,2?

Elev 1: Ja, jeg får det forkortet. Får du inn 1120? Det virker som det er forkortet på noe vis. Men 11,2 er det samme som, ja du skjønner ... (Gruppe 1)

Eleven har i dette tilfellet gjort det riktig, men observerer at svaret blir feil. Eleven ser likevel ikke hva som er problemet, og går ikke tilbake til formelen for å rette opp i feilen.

I flere tilfeller i datamaterialet finner vi utsagn som kan tyde på at det var en begrensende faktor at elevene ikke fikk benyttet algoritmen på riktig måte. Det ser ut å være en tendens at elevene mangler forståelse for hvordan algoritmen fungerer, og hva de ulike variablene og konstantene i formelen betyr. I flere av disse tilfellene står elevene fast fordi løsningen deres ikke er korrekt eller gir mening for dem ut fra ordlyden i oppgaven. En observasjon jeg har gjort meg i flere tilfeller er at elevenes misoppfatninger knyttet til algoritmen ofte kan løses av at elevene gjør en ny vurdering, og tester eller debugger koden sin for å se etter feil. Et godt eksempel på dette er utsagn 8, hvor eleven har gjennomført algoritmen riktig, men ikke har lagt merke til at det ene tallet mangler et siffer. Elevenes misoppfatninger innenfor algoritmebehandling er i mange tilfeller knyttet opp mot matematikk, noe som kan tyde på at den matematiske kunnskapen til elevene er et kjennetegn på misoppfatningene som oppstår i denne studien.

4.3 Strategisk kunnskap

De siste tre hovedkomponentene innenfor algoritmisk tenkning har jeg valgt å plassere innenfor «strategisk kunnskap». Utsagnene som er plassert innenfor disse tre underkategoriene bærer preg av større og dypere utfordringer enn i de foregående kategoriene, og retter fokus på hvordan den syntaktiske og konseptuelle kunnskapen blir anvendt. I «strategisk kunnskap» finner vi misoppfatninger som kjennetegnes av at elevene mangler gode og veletablerte strategier, samt at de har problemer med å teste eller forklare sitt eget arbeid.

4.3.1 Evaluering

I kategorien «evaluering» blir det lagt frem utsagn hvor elevene står fast i oppgaven, og kunne ha testet eller debugget koden sin for å løse problemet. I mange tilfeller oppleves det som at elevene ikke evaluerer arbeidet sitt hvis det blir feil, noe funnene i denne kategorien vil vise.

Utsagn 11

Lærer: Hva var renten din på da, elev 1?

Elev 1: 1200

Lærer: Renten. Skal vi se (går til skjerm)

Elev 3: Hvordan ... Ehm ... jeg får ikke frem svaret ...

Elev 1: Ja, jeg skjønner ... Nei, for at det ... Det er noe her som ikke ...

Lærer: Hvor mye vil vi sette inn? (Peker på skjerm). Etter at de har stilt spørsmål om det, så setter den innskudd til å være svaret på det spørsmålet der, ikke sant? I tillegg til at den setter utregning rente til å være svaret på det spørsmålet, ganger ingenting delt på hundre ... (Gruppe 1)

Utsagn 12

Elev 2(1): Må bare sjekke om det blir riktig her ... For jeg mener det blir riktig. Den formelen fire ganger hundre delt på 28 000, men svaret blir jo ikke riktig ... (Gruppe 1)

I utsagn 11 står elevene fast med oppgaven, fordi resultatet ikke kom frem på skjermen til høyre i scratch. I denne situasjonen blir læreren nødt til å gå gjennom programmet, eller debugge det. Også i utsagn 12 står eleven fast. I begge disse eksemplene kunne elevene ha valgt å debugge programmet selv, for å finne ut hvordan problemet kunne løses. Elevene ser etterpå hva som er problemet, fordi læreren påpeker at formelen er feil ved å lese den høyt. I denne situasjonen oppstår det en misoppfatning fordi elevene ikke ser at en liten korrigering i koden kunne ha gitt de riktig resultat.

Tilsvarende utsagn finnes gjentatte ganger i datamaterialet, og kan tyde på at elevene ikke har lært seg å evaluere sine egne løsninger. Elevene observerer ofte at det som blir gjort ikke gir de en fornuftig løsning på oppgaven, men benytter seg likevel ikke av å teste ut programmet eller være kritisk til koden de har laget. I de fleste tilfellene hvor dette har oppstått, har testing eller debugging av koden kunnet hjelpe med å komme frem til en mer relevant og hensiktsmessig løsning. Av disse grunnene er det derfor relevant å vurdere om manglende evner i å evaluere arbeidet kan føre til misoppfatninger i programmering. Om det er riktig vurdering, vil det bety at evnene elevene har i å evaluere har påvirkning på deres strategiske kunnskap.

4.3.2 Dekomponering

I denne kategorien retter jeg søkelyset mot misoppfatninger som har oppstått i elevenes forsøk på, eller fravær av, å dekomponere elementer i koden. Gjennomgående i datamaterialet er det lite eller ingen funn som tyder på at elevene har forsøkt, men ikke fått til å dekomponere problemet. Det er også få funn som tyder på at elevene skulle ha dekomponert, men ikke har gjort det. Av den grunn har jeg valgt å ta med et utsagn fra læreren, da det er antydning til at elevene skulle ha benyttet seg av dekomponering.

Utsagn 13

Lærer: Så vi trenger egentlig ikke å lage en formel for å sette inn noe i listen. Vi trenger bare å vite hva det er vi vil ha inn i lista. Og det vi vil ha inn i lista er jo egentlig det dere nettopp har laget. Innskuddet og gevinsten, ikke sant? Det vi kan gjøre da, hvis dere ser på de forskjellige elementene på mørkeoransje der ... Hvilken er det som er enklest å legge inn for å få lagt til noe som helst i en liste? (Gruppe 2)

I utsagn 13 ser vi at læreren gir instrukser til elevene hvor det blir påpekt at de ikke trenger å benytte seg av en hel formel, men bryte den opp for å se hva de egentlig har behov for. De kan dekomponere formelen for å finne en enklere løsning, og benytte de mindre delene for å løse andre problemer i oppgaven.

I arbeidet med oppgaven var det ikke mange eksempler på utsagn hvor elevene dekomponerte. Det er heller ikke noen spesifikke eksempler på at elevene forsøkte å dekomponere, men gjorde det feil. Det er derimot deler av datamaterialet som tyder på at elevene skulle ha benyttet seg av dekomponering for å løse oppgaven på en enklere måte. Selv om dette ikke peker på én konkret misoppfattelse, kan det være av betydning for vurderingen av elevenes evner til å dekomponere løsningene sine. Siden datamaterialet tyder på at elevene i liten grad benytter seg av dekomponering, kan det tilsi at dette er kjennetegn på misoppfatninger elevene har hatt i arbeidet med oppgaven.

4.3.3 Generalisering

I den siste underkategorien ser jeg på elevenes misoppfatninger når de har forsøkt å se mønstre og sammenhenger mellom ulike elementer. Her vil jeg legge frem eksempler hvor det har oppstått misoppfatninger knyttet til generalisering, for eksempel ved at de overgeneraliserer, eller ved at generaliseringen de har gjort ikke er gyldig.

Utsagn 14

Elev 1: Skal vi se ... Da bare skriver jeg ... Innskudd. Og så er det fortsatt delt på fire, er det ikke det da?

Lærer: Hvis renten er fire, så ja.

Elev 2: Men det er jo det, vi vet jo ikke renten. (Gruppe 1)

I utsagn 14 kommer det frem at elev 1 ønsker å benytte formelen de lagde tidligere i oppgaven, som gjaldt for et konkret eksempel. For å få en hensiktsmessig løsning på problemet, må eleven i dette tilfellet egentlig generalisere formelen for at den kan gjelde på alle eksempler uavhengig av tall. Siden eleven ikke ser dette selv, kan det tyde på at h*n har en misoppfatning knyttet til det å generalisere.

Utsagn 15

Lærer: Hvordan går det?

Elev 1: Mmm, nei vi tenker haha

Lærer: Hva er det som er annerledes her, i forhold til det dere nettopp gjorde?

Elev 2: Vi kan ikke bruke denne her, altså den grønne (peker på formelen de nettopp brukte, med tall inni), eller vi kan jo sikkert bruke den på en måte, men-

Elev 1: Den er ukjent

Elev 2: Ja, den er ukjent (Gruppe 1)

I dette utsagnet antyder elevene først at de ikke kan benytte seg av den konkrete formelen de lagde i starten av oppgaven. Deretter sier de at de kan bruke den «på en måte», men det virker som at elevene ikke forstår hvordan. De ser ikke at konstantene de har kommet med i første løsning kan byttes ut til å være variabler, og at de på denne måten hadde fått generalisert den. De får med andre ord ikke generalisert formelen ut fra tidligere løsning.

Utsagn 16

Lærer: Så det som kan være lurt å tenke på da, er hvordan vi skal klare å frem dette på skjermen her

Elev 2: Bakgrunn-nummer ...

Elev 1: Ehm, man må jo sette inn noe her, vent da ...

Elev 2: Ja

Elev 1: Gjenta for alltid? (Gruppe 1)

I utsagn 16 ser vi at elev 1 foreslår å benytte seg av en løkke, altså «gjenta for alltid»-funksjonen i Scratch. Elevens formål med løkken er å få svaret på formelen de har lagt inn, til å vises på skjermen gjennom hele koden. Én misoppfatning eleven har her, er knyttet til løkker og hvordan løkker fungerer i programmeringskontekst. En annen misoppfatning er at eleven forsøker å generalisere ved å gjenta formelen for alltid, noe som fører til en overgeneralisering.

De ulike gruppene elever (gruppe 1 og gruppe 2) viste relativt stor forskjell når det kom til å generalisere. Utsagnene ovenfor er alle tatt fra arbeidet til gruppe 1. Denne gruppen fikk også mindre instruksjoner i starten, og måtte finne formelen for rente selv. Av den grunn kan datamaterialet tyde på at disse elevene fikk mer problemer med å generalisere og forstå formelen som var nødvendig for å løse oppgaven. Det vi ser i utsagnene er at elevene i flere tilfeller gjør et forsøk på å generalisere, men at løsningen ikke er tilfredsstillende i henhold til oppgavens formål. I det siste utsagnet, som jeg klassifiserer som «overgeneralisering», forsøker eleven å finne en løsning som gjør at den gjelder for hele programmet, men løsningen tyder på at det er mangel på forståelse i hvordan selve programmet fungerer.

4.4 Veiledende instruksjoner

Veiledende instruksjoner ble en egen kategori da jeg i forbindelse med analyse og koding av datamaterialet, ble oppmerksom på hvor stor rolle læreren spilte for at elevene skulle komme seg frem til en hensiktsmessig løsning. I dette underkapitlet finner vi derfor utelukkende utsagn hvor læreren har hatt en markant rolle. Hensikten med dette er å få frem i hvilke situasjoner elevene har stått fast eller misoppfattet deler av oppgaven, og i hvilke situasjoner læreren har vært nødt til å spesifisere fremgangsmetoden for at elevene skulle komme seg videre i arbeidet. I denne kategorien finner vi utsagn som passer for omtrent alle konseptene i algoritmisk tenkning. I underkategorien «abstrahering» er det derimot ikke gjort noen funn som kan være relevante. Av den grunn velger jeg å dele denne kategorien inn i fem av de seks komponentene til algoritmisk tenkning, hvor underkategorien «abstrahering» ikke er inkludert. For hver underkategori er det utsagn med en tilhørende kort beskrivelse. I slutten av kategorien «veiledende instruksjoner» vil det være en oppsummerende beskrivelse av utsagnene, som setter søkelys på hvilken rolle

læreren har hatt og hvilke antakelser en kan gjøre om elevenes forståelse/misoppfatninger på bakgrunn av disse utsagnene.

4.4.1 Automatisering

Under automatisering finner vi utsagn som kan tyde på at læreren har gitt elevene tilbakemeldinger som har hjulpet de i å automatisere koden sin ytterligere. Det vil si at elevene har fått hjelp til å forstå hvor man skal finne riktige elementer til koden sin, fordi de ikke har funnet det selv.

> **Utsagn 17**
>
> *Lærer: Mm, hvis dere vil legge noe til i gevinst-lista? Hva er mest logisk å legge til i den listen?*
>
> *Elev 1: Var det ikke ...*
>
> *Lærer: Vi skal jo egentlig lage en liste over gevinst. Så hvilken variabel kan vi legge til i gevinst-lista? (Gruppe 1)*

I dette utsagnet er læreren nødt til å stille elevene et konkret spørsmål for at elevene skal forstå hvordan de skal få frem gevinsten de har regnet ut i listen over gevinst for hvert år. Spørsmålet læreren stiller kan tolkes som et ledende spørsmål, og gir ikke rom for at eleven skal tenke selv. Dette spørsmålet kom i etterkant av at elevene lenge hadde prøvd å forstå hva de skulle legge i listen over gevinst, og ble derfor sett på som nødvendig for at elevene skulle komme seg videre i oppgaven.

> **Utsagn 18**
>
> *Lærer: Hvis dere scroller litt lengre ned på variabelen der nå, så ser dere at det er en egen en som heter ... Ja, nytt navn på liste. Så da kan dere ha en egen liste som heter «Gevinst», der dere vil få en liste over gevinsten for hvert år. Slik som 1120, det vil være gevinsten for år én. I år to vil den øke litt, og i år tre vil den øke litt mer. Det er listen over gevinst. (Gruppe 2)*

Også i dette utsagnet ser vi at læreren spesifikt henviser til hva som er naturlig å gjøre for å komme seg videre i oppgaven. Det virker som elevene ikke har forstått hva som egentlig må gjøres for å kunne lage en løsning på problemet, og i dette tilfellet tyder det på at mangel på øvelse i det aktuelle språket (Scratch) stopper elevene fra automatisere løsningene sine selv.

> **Utsagn 19**
>
> *Lærer: Slike typer blokker som har med matematikk å gjøre, er på den grønne som heter «operator». Der finner dere det meste som har med matematikk å gjøre hvert fall (Gruppe 2)*

I dette utsagnet gir læreren beskjed om hvor elevene kan finne enkelte elementer i scratch. Dette kan tyde på at elevene ikke er kjente med Scratch som programmeringsspråk. I denne situasjonen visste elevene hva de trengte, men trengte hjelp til å finne riktige blokker.

4.4.2 Algoritmebehandling

I dette underkapitlet ser vi på utsagn hvor det kommer frem at læreren har bidratt til å få elevene til å bedrive algoritmebehandling på en hensiktsmessig måte. Her vil det komme frem indirekte spørsmål som skal få elevene til å tenke selv, men også noen forklaringer som gjør at elevene forstår algoritmen de skal behandle.

Utsagn 20

Lærer: Men dere har hvert fall innskudd, og dere har rente. Og så trenger dere å kalkulere det. Så hva var formelen? (Gruppe 1)

I dette utsagnet oppsummerer læreren til elevene hva slags informasjon de har, og forteller dem at de kan kalkulere løsningen sin ved hjelp av den informasjonen. Det læreren egentlig antyder er at de må benytte seg av tidligere løsning, altså den formelen som gjaldt for bare ett tilfelle, slik at de får lagt inn informasjonen de har funnet ut.

Utsagn 21

Lærer: «Hvor mange år?», jo, det var la oss si 15 år da. Så hvis svaret er 15 så vil dette skje 15 ganger. Da får du en liste her med 15 år, kan du si. Så da passer det jo at denne kommer etter det spørsmålet her. Og så, innskudd ... Hvordan kan du få satt innskuddet til å bli svaret da (svaret på spørsmål om innskudd)? (Gruppe 2)

I dette utsagnet står elevene fast fordi de er usikre på hvordan de skal legge til noe i en liste. Den første listen har de nettopp laget, så læreren går gjennom koden muntlig, og forteller hva som kommer til å skje ved å bruke et tilfeldig eksempel. Deretter hinter læreren om at denne fremgangsmetoden kan brukes på et tilsvarende problem videre i koden.

Utsagn 22

Lærer: Dere kan jo, i stedet for å kalle det for x, y og z, velge å kalle det for år, krone og rente også. For det har jo egentlig ingen betydning, ikke sant? Det står slik (i oppgaveteksten) bare for å vise at det kan være hva som helst. Det er derfor jeg har brukt de bokstavene. (Gruppe 1)

Elevene prøvde å lage en formel i Scratch som tok utgangspunkt i x, y og z. Elevene brukte tid på å lete etter blokker som hadde x, y eller z i seg. I dette tilfellet måtte læreren derfor spesifisere at variablene i formelen kan endres, og at det ikke er fastsatt at de må hete x, y og z. Dette kan tyde på at elevene ikke er helt fortrolig med formelen for rente, og at de ikke vet hvordan de kan benytte algoritmen på hensiktsmessig måte i koden sin.

Utsagn 23

Lærer: Det er ikke prosent, men tallet. Hvis det er fire prosent, så er det tallet fire. Det er derfor det blir fire delt på hundre. Det er egentlig det samme som det du setter inn, ganger prosenten, fire prosent. (Gruppe 2)

I dette utsagnet må læreren forklare til elevene hvordan formelen er bygd opp. Elevene hadde kommet frem til en formel som kunne regne ut 4% av 28 000, men hadde problemer med å forstå hva de ulike elementene i formelen betydde. Av den grunn fikk de problemer med å legge inn formelen i koden, fordi de ikke visste hvilken variabel som skulle plasseres hvor.

Utsagn 24

Lærer: Jeg kan jo forklare hva det er. Rente er noe som har med økonomi å gjøre. Hvis man for eksempel setter inn en viss sum på en bankkonto, så har man en rente på den bankkontoen. Og den renten kommer i prosent. Det vil si at hvis du har 4% rente, slik som i oppgaven, så er det 4% av den summen du har inne på bankkontoen. Hvis du setter inn ... Si at du setter inn 100kr på en bankkonto- (lærer blir avbrutt) ... så får du 4% rente av å ha de pengene inne på kontoen. Så da vil du jo få fire kroner, hvis du har 100kr inne på kontoen. For da er fire prosent av hundre lik 4kr, ikke sant. Så vil det stige etter hvert, mm. (Gruppe 2)

Før elevene fikk oppgaven utdelt visste de ikke hva rente var, og av den grunn blir det i utsagn 24 forklart hva som menes med rente. Dette var nødvendig for læreren slik at elevene skulle vite hva slags matematisk konsept de arbeidet med.

4.4.3 Evaluering

Under «evaluering» i kategorien «Veiledende instruksjoner» finner vi utsagn som antyder at læreren har vært nødt til å teste eller debugge koden til elevene når de har stått fast, fordi de ikke har gjort det selv. I tillegg finner vi utsagn hvor læreren ber elevene om å planlegge før de begynner å programmere i Scratch.

Utsagn 25

Lærer: «Hvor mange år?», jo, det var la oss si 15 år da. Så hvis svaret er 15 så vil dette skje 15 ganger. Da får du en liste her med 15 år, kan du si. Så da passer det jo at denne kommer etter det spørsmålet her. Og så, innskudd ... Hvordan kan du få satt innskuddet til å bli svaret da (svaret på spørsmål om innskudd)? (Gruppe 2)

Læreren går gjennom (tester) hva som kommer til å skje i koden muntlig, og forteller til elevene hvilken løsning det vil gi. Elevene hadde på dette tidspunktet ikke testet koden sin selv, eller prøvd den med tilfeldige tall. Av den grunn var de usikre på hva som skjedde med listen, og fikk ikke kommet seg videre i oppgaven. At læreren er nødt til å teste koden for elevene, kan tyde på at elevene ikke så behovet for dette selv, noe som i dette tilfellet førte til at elevene ikke forsto hva de skulle gjøre videre.

Utsagn 26

Lærer: Det er et godt poeng. Det er en god start det, elev 1, at den må kunne gange og dele. Da trenger dere kanskje noen blokker i scratch som kan å gange og dele? Jeg anbefaler dere kanskje å lage en ... Diskuter litt. Hvordan er det denne kalkulatoren burde være? Slik som at den bør kunne gange og dele, og så eventuelt tegne det som en skisse på arket deres før dere setter i gang på scratch. (Gruppe 2)

I dette utsagnet ser vi at læreren spesifikt forteller elevene at de bør planlegge koden sin, og lage en skisse over hvordan de ser den for seg før de lager den i Scratch. Dette sto også i oppgavearket elevene fikk delt ut, men det var likevel behov for å spesifisere det til elevene når de satte i gang med arbeidet.

Utsagn 27

Lærer: Du kan jo prøve å endre tallene inni der (formelen) og se hva som skjer? Hvis du sier at du for eksempel skal sette inn 40 000 på den kontoen da. (Gruppe 2)

Utsagn 28

Lærer: Hvis dere tar ... nå kan dere jo bestemme selv da. Lat som dere har vunnet i lotto eller fått en arv eller hva som helst, og at renten er på en prosent mellom 1 og 5 da, kan vi si. Så kan dere sjekke ut om programmet deres tar det opp og om dere får regnet det ut. (Gruppe 1)

I både utsagn 27 og 28 ser vi at læreren veileder elevene til å teste ut programmet sitt med tilfeldige tall. Dette ble gjort for å få elevene til å sjekke om det fungerte på andre eksempler enn det de hadde brukt, og om løsningen de kom frem til var egnet for problemet. Slike eksempler finnes det også flere av i datamaterialet, noe som tyder på at elevene selv ikke ser behovet for å teste løsningene sine.

4.4.4 Dekomponering

Under dekomponering finner vi et utsagn som retter søkelys på lærerens rolle i en situasjon hvor elevene hadde behov for å dekomponere koden sin.

Utsagn 29

Lærer: (peker på operasjonene, grønn). Ja, og her har dere innskudd, og her har dere rente, delt på hundre. Så hva kan dere gjøre for å få innskudd ganger rente delt på hundre da, inne i den operatoren der?

Elev 1: Ehm ... Lage variabler?

Lærer: Ja, men hvis dere ser på variablene. Du har jo allerede noen variabler der. (Gruppe 1)

I dette utsagnet prøver læreren å veilede elevene til å finne ut hvilken komponent de kan benytte seg av inne i formelen, for å få formelen de trenger når de skal løse oppgaven. Tidligere hadde elevene bestemt variablene «innskudd» og «rente», og formålet var derfor at de skulle benytte variablene inne i formelen. I stedet for å benytte samme variabler som tidligere, foreslår elev 1 å lage nye variabler. I denne situasjonen kan man derfor anta at eleven ikke ser formålet med å dekomponere. Hadde eleven hatt forståelse for dette ville h*n ha sett at de mindre delene de lagde tidligere, kunne passet inn i en større del av oppgaven.

4.4.5 Generalisering

I dette underkapitlet ser vi på utsagn hvor læreren forsøker å formulere oppgaven, slik at elevene skal kunne generalisere løsningen sin.

Utsagn 30

Lærer: Så vi kan si det, at vi vet ikke renten, vi vet ikke innskuddet. Hundre vet vi, for det er jo i formelen, den varierer ikke. Så hvis dere får til å lage en kode som gjør at vi kan legge inn tilfeldige tall på de to der, innskudd og rente, så har vi kommet ganske langt egentlig. (Gruppe 1)

Utsagn 31

Lærer: Og da skal det være tilfeldige tall vi skal legge inn. Det trenger ikke å være akkurat 28 000 og 4 i rente. Det kan være et hvilket som helst tall. Og så tror jeg jeg skrev noen tips på baksiden der også. Så det kan dere jo også se på. For formålet med programmet her er jo at man skal ha anledning til å legge inn hvilke tall man vil, uten å gå inn i programmet for å endre variabelen hver gang. (Gruppe 1)

Utsagn 32

Lærer: Enn hvis dere ikke vet da ... Eller. Si at dere skal lage en kalkulator til en som jobber i banken. Og så kommer en av dere inn og sier «Jeg har fått noen penger i arv fra farfar etter at han gikk bort, får du til å beregne hvor mye jeg får inne på denne kontoen for meg?». Prøv å lage en kalkulator som kan bestemme ut ifra hvilken sum som helst, og antall år som helst, hvor mye du vil tjene på den renten. (Gruppe 1)

I utsagn 30, 31 og 32 kommer det frem at læreren benytter ulike språklige metoder for å få elevene til å generalisere formelen de skal frem til i oppgaven. Læreren forsøker å tipse elevene om å se det generelle i formelen, men må benytte ulike forklaringer for at elevene skal forstå hvordan de skal gå frem.

4.4.6 Hva tyder de veiledende instruksjonene på

I kategorien «Veiledende instruksjoner» er det flere eksempler på områder hvor elevene har hatt behov for korrigering eller veiledning fra læreren. Ingen kategorier ser ut til å utmerke seg særlig mer enn andre, men «abstrahering» og «dekomponering» utmerker seg i den grad at de ikke/sjeldent oppsto i datamaterialet. En grunn for dette kan være at abstraksjon og dekomponering ikke kommer like tydelig frem, fordi prosessen bak elevenes fremgangsmetoder ikke vises i transkripsjonen. Det kan også være fordi ordlyden i oppgaven ikke ga rom for å abstrahere eller dekomponere i like stor grad som de andre komponentene i algoritmisk tenkning. Lærerens rolle for at elevene skulle komme seg videre i oppgaven kan virke å være stor basert på utsagnene i denne kategorien, noe som kan tyde på at elevene har hatt en del misoppfatninger i arbeidet med Scratch som programmeringsspråk. Denne kategorien bidrar til å forsterke tolkningen som ble gjort i kapittel 4.1-4.3.

5 Diskusjon

Den norske skolen har over lengre tid forsøkt å implementere programmering og algoritmisk tenkning som en del av barnas målsetninger i skoleløpet (Sevik, 2016), men hvordan det skal gjøres og hvilke pedagogiske forutsetninger det krever kommer ikke like tydelig frem. I undervisning hvor formålet er at elevene skal benytte seg av, eller lære om, programmering som verktøy, er det en forutsetning at læreren har kunnskap om hvilke metoder som fungerer og ikke fungerer for at elevene skal få best mulig utbytte. I denne studien er formålet å sette søkelys på hva som kjennetegner 10. klasse elevers misoppfatninger i arbeid med matematiske oppgaver i et blokkprogrammerings-miljø, og en personlig målsetting er å benytte resultatene av studien som en rettesnor i min fremtidige arbeidshverdag.

En overveiende tendens fra den innhentede empirien presentert i kapittel 4.0, gir oss et innblikk i omfanget av misoppfatninger som kan oppstå under 10. klasse elevers matematiske arbeid i programmeringsspråket Scratch. I dette kapittelet vil jeg ta for meg funnene som ble analysert, og belyse disse ved bruk av relevant teori. Organiseringen av kapittelet er gjort med samme utgangspunkt som kapittel 4.0, hvor jeg arrangerer de tre hovedkategoriene som egne underoverskrifter. Jeg vil i kapittel 5.1-5.3 diskutere misoppfatninger som har oppstått i de tre kategoriene, og drøfte hvorvidt disse synspunktene har en reell forankring i virkeligheten. Videre vil jeg diskutere mulige årsaker knyttet til funnene som er gjort, og vurdere om det kan forekomme andre ringvirkninger som følge av disse.

Min metode for å besvare problemstillingen «Hva kjennetegner 10. klasse elevers misoppfatninger i arbeid med matematikk i et blokkprogrammeringsmiljø?», har som beskrevet i kapittel 3.0 vært å ta utgangspunkt i kunnskaper som legges spesielt vekt på når det snakkes om programmering i skolen. Av ulik litteratur ser vi at «algoritmisk tenkning» er et begrep som går igjen, noe som også gjenspeiles i læreplanen (Utdanningsdirektoratet, 2020). Av den grunn ser jeg det både som naturlig og formålstjenlig for oppgaven, at problemstillingen vil bli drøftet i lys av de definisjonene som finnes for begrepet algoritmisk tenkning. Tanken med dette kapittelet er derfor at leseren skal se på resultatene som flere deler av et større «bilde» (algoritmisk tenkning), og at kategoriseringen av begrepet «algoritmisk tenkning» kan kaste et lys på hvilke områder misoppfatningene har vært fremtredende på, i min studie.

Før jeg går videre i diskusjonen, vil jeg gjøre en oppsummering av de viktigste funnene fra kapittel 4.0. Det som kommer tydelig frem er at det underveis i arbeidet med matematikk i Scratch, har oppstått problemer knyttet til forskjellige faktorer ved algoritmisk tenkning. Elevenes utsagn belyser situasjoner hvor det har oppstått misforståelser eller misoppfatninger knyttet til problemløsningen, og disse misoppfatningene finner vi i samtlige komponenter fra algoritmisk tenkning. Dette kan tyde på at elevenes misoppfatninger ikke bunner i noen få, spesifikke områder, men at misoppfatningene oppstår gradvis og i forskjellige omfang gjennom arbeidet med problemet. Disse antydningene blir igjen bekreftet i kapittel 4.4 Veiledende instruksjoner, der det blir tydelig hvor stor rolle læreren spilte i de ulike områdene elevene sto fast på.

5.1 Misoppfatninger syntaktisk kunnskap

Å overføre en tanke, en prosess eller en løsning av et problem inn i et digitalt språk, krever at en har forståelse for språket og hvordan det er oppbygd. Dette omtaler Qian & Lehman (2017) som syntaktisk kunnskap. Rollen syntaktisk kunnskap har i arbeid med programmering er derfor stor, da denne delen av prosessen er kritisk for å faktisk evne å skape en kode som fungerer for sitt formål. I arbeid med programmering i skolen, blir gjerne mindre komplekse programmeringsspråk benyttet. Med mindre komplekse programmeringsspråk siktes det til blokkprogrammeringsspråk, slik som Scratch eller muligens det mer kjente «Microbit». Det finnes også andre typer språk som benyttes i skolen, og noen lærebøker eksemplifiserer oppgavene sine ved hjelp av Python (Kongsnes & Wallace, 2020). Likheten mellom disse språkene er at de alle har unike måter å bygge opp programmene sine på. Python er et tekstbasert programmeringsspråk, hvor det som utføres i koden er skrevet med ord av brukeren. Microbit er basert på grafiske blokker, men har en annen oppbygging og utforming enn Scratch. I Microbit har man i tillegg muligheter til å overføre koden til en fysisk mottaker, som kan plasseres på ulikt utstyr slik at programmeringen kommer til syne fysisk. Med andre ord kan elevene møte på mange programmeringsspråk i løpet av skolegangen, og må dermed ha kunnskaper om mange ulike funksjoner og oppbygginger.

Innledningsvis i arbeidet med oppgaven elevene fikk utdelt, var et av spørsmålene som ble stilt til elevene: «Har du jobbet med Scratch før?». Sett bort fra den ene eleven i gruppe 2 som hadde hatt programmering som valgfag tidligere, var det ingen av elevene som hadde benyttet scratch til å løse oppgaver tidligere. Flere av elevene hadde derimot prøvd Microbit tidligere, og hadde en viss kjennskap til hvordan Microbit sitt språk var bygd opp. Uten å påstå at denne faktoren var særskilt avgjørende for elevenes suksess i arbeidet med det tekniske i Scratch, er det likevel et poeng som har behov for å diskuteres. Å kunne «automatisere» arbeidet sitt er en del av algoritmisk tenkning (Gjøvik & Torkildsen, 2019), og knyttes opp til den syntaktiske kunnskapen fordi det dreier seg om å foreta seg noe teknisk, og å forstå syntaksen. Å ikke vite eller forstå hvordan en skal foreta seg noe teknisk/syntaktisk i et program, vil naturlig nok få konsekvenser for hvordan arbeidet i programmet utspiller seg. Slik som Qian & Lehman (2017) beskriver, er det å ikke være kjent med programmeringsspråket en faktor som kan skape misoppfatninger. I funnene som ble gjort i kapittel 4.1.1 Automatisering, og 4.4.1 Automatisering, ser vi at elevenes kunnskaper i Scratch påvirker effektiviteten og problemløsningen deres.

I underkategorien «4.1.1 Automatisering», gir funnene oss et bilde på hvordan elevene arbeidet med oppgaven i Scratch. Utsagn 1 og 2 tyder på at det var vanskelig for elevene å finne en løsning for å sette sammen forslagene de hadde laget i koden. Dette hadde konsekvenser for elevene i form av at de ikke fikk prøvd ut løsningen sin, i tillegg til at det videre arbeidet ble satt på vent. I utsagn 3 ser vi at eleven ønsker å benytte seg av en funksjon i Scratch som egentlig ikke er nødvendig for oppgaven. Det er rom for å spekulere i om dette kan skyldes elevens naturlige språk, slik som Qian & Lehman (2017) forklarer det. Funksjonen eleven ønsker å bruke er det som kalles «vilkår», og er i Scratch formulert som «hvis/ellers». Det er mulig elevens naturlige oppfatning av ordene «hvis/ellers» skaper en forvirring av hvordan denne funksjonen kan brukes, og i dette tilfellet bidrar det til å skape en misoppfatning når eleven skal automatisere oppgaven. I tillegg til dette, kan også oppgavens kompleksitet eller kognitive krav spille en rolle for hvordan elevene arbeider med det syntaktiske i programmering (Qian & Lehman, 2017). Når oppgaven blir for «stor», er det enklere for elevene å glemme enkle og elementære deler av programmeringen, fordi den totale belastningen gjør at små elementer blir glemt.

Oppgaven elevene fikk utdelt startet relativt enkelt, men etter hvert ble oppgaven mer kompleks i form av at matematikken måtte generaliseres, og nye funksjoner i programmet måtte tilføres for å få en hensiktsmessig løsning. Dette kan ha ført til at elevene glemte enkle deler av programmeringens syntaks som de i utgangspunktet mestret, og av den grunn ledet dette til misoppfatninger.

I hovedkategorien «Veiledende instruksjoner», er utsagnene som kommer fra læreren beskrivelser av hvordan elevene skal kunne overføre tankene de har inn i Scratch. Siden «automatisering» dreier seg om å bringe løsningen inn i et programmeringsspråk (Gjøvig & Torkildsen, 2019), er det mulig å anta at elevenes og lærerens utsagn knyttet til automatisering, kan anses som misoppfatninger. Når læreren i disse utsagnene instruerer elevene i arbeidet med å løse problemet, baseres det på at elevene på forhånd har stått fast og ikke har kommet frem til en løsning på egen hånd. I mange tilfeller dreier ikke misoppfatningene seg om mangel på forståelse for matematikk eller programmering generelt. De misoppfatningene elevene har hatt når de har forsøkt å overføre koden sin inn i programmeringsspråket, ser ut til å preges av at elevene ikke er godt kjent med språket eller vet hvordan de skal benytte seg av de ulike funksjonene i språket. Dette er en faktor som kan føre til misoppfatninger, i følge Qian & Lehman´s (2017) studie, og det er derfor mulig å se på dette som et kjennetegn på elevenes misoppfatninger i denne casen.

På en annen side trenger ikke misoppfatninger rundt det å automatisere koden å bety at elevene ikke er i stand til å programmere, eller at de ikke forstår hvordan programmering fungerer. I likhet med Qian & Lehman´s (2017) funn, er det også i denne studien andre faktorer som muligens spiller større rolle i forståelsen av programmering. Å slite med syntaks eller tekniske aspekter ved programmering beskrives som «overfladiske» misoppfatninger, som lett kan observeres og gjøres noe med (Qian & Lehman, 2017). Dette er en motsetning til de misoppfatningene vi finner i konseptuell og strategisk kunnskap, noe jeg kommer tilbake til i kapittel 5.2 og 5.3.

5.2 Misoppfatninger konseptuell kunnskap

Misoppfatninger knyttet til den konseptuelle kunnskapen elevene innehar, er i større grad basert på dypere og mer signifikante feil (Qian & Lehman, 2017). Innenfor konseptuell kunnskap finner vi «abstrahering», som innebærer å være i stand til å ta bort irrelevante detaljer, for å skape en enklest mulig kode (Czismadia et al., 2015). I analysen av datamaterialet blir det påpekt at misoppfatninger innenfor denne delen av algoritmisk tenkning, ikke forekom ofte i transkripsjonen. Det er grunn til å diskutere hvorvidt dette «ikke-funnet» er et funn i seg selv, og om det til tross for en lav forekomst i empirien, likevel kan anses som et kjennetegn på elevenes misoppfatninger i arbeid med matematikk i et blokkprogrammeringsmiljø.

Fra elevenes formuleringer ble det kun trukket frem ett spesifikt eksempel, og i kategorien «Veiledende instruksjoner» er det ingen funn som åpenbart tydet på misoppfatning innenfor abstrahering. I kapittel 4.2.1 Abstrahering, ser vi at eleven gjør det motsatte av å abstrahere, altså å legge til unødvendig informasjon i koden. Dette funnet gjør at abstrahering kan virke vanskelig for eleven, samtidig som grunnlaget for å antyde dette ikke er stort. Det som derimot kan sies om dette tilfellet, er at elevens strategi for å løse problemet er ukomplett, og tyder på at mangelen på mentale modeller (Qian & Lehman, 2017) forårsaker misoppfatningen i dette utsagnet. Cui & Ng (2021) påpeker også at elevene kan ha misoppfatninger knyttet til abstraksjon, da det ofte forekommer eksempler på at elevene ikke evner å lage et program uten å måtte gjøre prosessene lange og

vanskelige for seg selv. Dette ser vi spesifikt i utsagn 7, men er ikke gjentagende for resten av datamaterialet. Det blir derfor vanskelig å konkludere med at misoppfatninger innenfor abstraksjon er noe som kjennetegner elevenes misoppfatninger.

Når det kommer til forekomsten av funn som kan tyde på misoppfatninger innenfor abstraksjon, er det verdt å diskutere om oppgavens ordlyd kan ha påvirket denne faktoren. På oppgavearket ser vi at problemet er utformet på en slik måte at elevene må følge en «oppskrift» stegvis. Formålet med å gjøre det slik var i utgangspunktet at elevene skulle se helheten i oppgaven, og bevisstgjøre elevene om at «steg 1» var viktig for å kunne gjøre «steg 4», for eksempel. Det oppgavearket også inneholder, er en liste over tips som anbefales å se på. Det denne formuleringen kan ha ført til, er at elevene ikke tenker utenfor boksen, og heller ikke har hatt behov for å abstrahere i arbeidet med koden. Dette fordi oppgaven har vært «gitt» dem på forhånd, og av den grunn har spørsmål om hva som skal være med innholdsmessig ikke vært et tema. En annen faktor som kan forklare den lave forekomsten er at elevenes abstraheringsprosess ikke kom frem i transkripsjonen. Det vil si at elevenes metode for å abstrahere ikke ble nevnt muntlig av elevene, men at de gjorde modifiseringer direkte i programmet, eller at løsningene deres ikke hadde behov for å abstraheres.

Den andre kategorien tilhørende konseptuell kunnskap, er algoritmebehandling. Algoritmebehandling er den delen i algoritmisk tenkning hvor man utfører og benytter seg av algoritmen på en hensiktsmessig måte (Czismadia et al., 2015). Dette forutsetter også at en vet hva algoritmen betyr, og får til å forklare den. I empirien som ligger til grunn for denne studien, er det flere funn som tyder på at dette området har vært en utfordring for elevene. I kapittel 4.2.2 Algoritmebehandling, ser vi at elevene har forskjellige misoppfatninger når det kommer til algoritmen og hvordan de skal behandle den. Under arbeidet med å innhente empiri, ble det tydelig at elevene som fikk utdelt første oppgaveark (Gruppe 1), brukte mye tid på å finne og forstå formelen de trengte for å løse problemet. Som konsekvens endret jeg oppgaveordlyden før Gruppe 2 skulle gjennomføre prosjektet, slik at mindre tid gikk bort til nettopp dette. Empirien bærer derfor preg av at dette ble et større problem i den ene gruppen, enn i den andre. Likevel er det grunn til å diskutere disse funnene, da gruppene som helhet ses på som en case i denne studien.

I utsagn 8 og 9 ser vi at elevene har vansker med å forstå hva slags betydning de ulike komponentene i formelen for å regne ut renten betyr. Det vi ser er at elevene ikke får til å rette opp programmet sitt, fordi forståelsen av formelen ikke er tilstrekkelig. Som elevenes faglærer nevnte i forkant av øktene som ble gjort, hadde ikke elevene arbeidet noe særlig med økonomi og rente i matematikk tidligere. Forkunnskapene de hadde i matematikk, kan derfor ha vært med å påvirke arbeidet deres i programmering. Dette er en faktor Qian & Lehman (2017) peker på som en mulig årsak til at det oppstår misoppfatninger i programmeringskontekst hos elevene. Det er av den grunn mulig at noen av misoppfatningene til elevene kjennetegnes av mangel på matematisk forståelse. I utsagn 10 ser vi at den ene eleven har fått riktig svar, men mangler en desimal i resultatet sitt. Eleven poengterer i utsagnet at det er «det samme» som det faktiske svaret, men det ser ikke ut til at eleven forstår hvordan h*n skal endre formelen slik at den gir riktig løsning. Av den grunn er det også i dette tilfellet mulig å anta at feilaktige forståelser og anvendelser av algoritmene er noe som kjennetegner misoppfatningene elevene har, som igjen kan ses i sammenheng med elevenes forkunnskaper (Qian & Lehman, 2017).

I kapittel 4.2.2 Algoritmebehandling, finner vi flere utsagn som peker på at elevene har hatt et behov for å få instruksjoner av læreren når de skulle bedrive algoritmebehandling.

I utsagn 24 ser vi at læreren har behov for å forklare hva rente egentlig betyr, noe som støtter opp antakelsen om at elevenes matematiske kunnskap har vært med å skape misoppfatninger i arbeidet med programmering. Også i de andre utsagnene ser vi at læreren har innvirkning på elevenes arbeid, i form av at læreren må forklare elevene hva formelen betyr, eller hvordan en kan benytte formelen. Det faktum at læreren er såpass deltakende i arbeidet med elevenes forståelse og anvendelse av algoritmene i koden, kan ses i sammenheng med at elevene har ukomplette mentale modeller knyttet til hvordan en skal arbeide med programmering. Lærerens behov for å presisere det matematiske/algoritmiske inneholder i koden, kan tyde på at elevenes tankegang er unøyaktige, og er også i tråd med resultatene til Qian & Lehman (2017).

5.3 Misoppfatninger strategisk kunnskap

I hovedkategorien strategisk kunnskap, finner vi dekomponering, evaluering og generalisering. Kjerneinnholdet i strategisk kunnskap, er hvordan den syntaktiske og konseptuelle kunnskapen blir benyttet sammen for å løse problemer i programmering (Qian & Lehman, 2017). Det vil si at misoppfatninger innenfor for eksempel automatisering (syntaktisk kunnskap), kan føre til misoppfatninger i evaluering (strategisk kunnskap). Det samme gjelder hvis man for eksempel har misoppfatninger innenfor algoritmebehandling (konseptuell kunnskap), så kan dette føre til misoppfatninger innenfor generalisering (strategisk kunnskap). I denne delen av diskusjonen vil jeg se på det som kjennetegner misoppfatningene til elevene i områdene som er kategorisert innenfor strategisk kunnskap.

I denne hovedkategorien finner vi blant annet «dekomponering», som vil si at man kan bryte opp et problem i mindre deler, og evner å forstå og utvikle disse delene separat fra hverandre (Czismadia et al., 2015). I kapittel 4.3.2 Dekomponering, kommer det frem at det i datamaterialet ikke inneholdt mange eksempler på misoppfatninger knyttet til dette området. Elevenes utsagn kunne ikke tyde på at dette var et gjentakende problem, men likevel trekkes det frem et eksempel på en situasjon hvor elevene *skulle ha* dekomponert. I utsagn 13 forklarer læreren til elevene at informasjonen de trenger allerede er i oppgaven, og at de kan benytte mindre deler av en tidligere kode til å fullføre arbeidet sitt. Dette utsagnet ses derfor på som et eksempel hvor elevene har misoppfattet hvordan og i hvilke situasjoner en skal dekomponere. Også i kapittel 4.4.4 Dekomponering, viser utsagn 29 at læreren har forsøkt å veilede elevene til å dekomponere underveis i arbeidet. Det kommer frem at elevene i dette tilfellet ikke forsto hva som skulle til for å skaffe den nødvendige informasjonen for å komme seg videre i oppgaven. Det er derfor klart at elevene trengte veiledning for å kunne dekomponere løsningen sin. Likevel er det ikke funn som tyder på at dette er gjentakende for denne casen.

Til tross for at lærerens veiledning i utsagn 13 og 29 kan tyde på misoppfatninger innenfor området dekomponering, er det derimot lite funn ellers i datamaterialet som kan bekrefte at dette kjennetegner misoppfatningene elevene hadde. I likhet med «abstraksjon», ser jeg også i dette tilfellet en grunn til å diskutere om årsaken til fraværet av funn, i denne casen kan være et kjennetegn på misoppfatninger i seg selv. En mulig årsak til at datamaterialet bærer lite preg av dekomponering, kan også i denne kategorien skyldes at transkripsjonen i seg selv ikke inneholder noen form for misoppfatninger innenfor dekomponering. Det vil si at elevene ikke har uttalt tydelig at de feilaktig har, eller har forsøkt, å dekomponere. En annen faktor som kan være relevant, er min rolle som lærer i situasjonen. I datamaterialet er det kun ett tilfelle hvor læreren forsøker å løfte frem dekomponering som en del av prosessen. Dette kan skyldes at oppgaven ikke ga rom for det, men det kan også skyldes mangel på kunnskap eller erfaring fra lærerens (min) side.

Dette er noe Qian & Lehman (2017) beskriver som en mulig årsak til misoppfatninger hos elevers arbeid med programmering. På en annen side er det vanskelig å vite om dette har hatt direkte påvirkning på elevenes misoppfatninger innenfor dette området. Av den grunn velger jeg å tolke det som at dekomponering ikke er et område som kjennetegner elevenes misoppfatninger i denne casen.

Den andre komponenten fra algoritmisk tenkning som vi finner i hovedkategorien strategisk kunnskap, er «evaluering». Dette området inneholder elevenes misoppfatninger knyttet til testing og debugging av koden (Czismadia et al., 2015). Den empiriske dataen gir oss et innblikk i situasjoner hvor elevene sto fast med oppgaven, og nevnt tidligere i dette kapitlet, har det vært ulike områder hvor disse problemene har oppstått. I flere av tilfellene hvor elevene har hatt problemer, har de i mange tilfeller ikke visst hva de skal gjøre. I disse tilfellene ser vi at læreren har vært nødt til å forklare til eleven hva som skjer i koden de har laget, for å få elevene oppmerksomme på eventuelle feil eller mangler. I utsagn 11 ser vi at eleven sliter med å få frem svaret på skjermen. Når læreren debugger koden til eleven muntlig, blir eleven oppmerksom på at formelen som er blitt brukt i en liten del av koden, ikke gir mening. I utsagn 12 har eleven kommet med en løsning, men eleven sier også i dette tilfellet at løsningen ikke blir riktig. Likevel ser vi i begge disse utsagnene at elevene ikke tester eller debugger koden sin for å finne feilen, til tross for at de er oppmerksomme på at det foreligger en feil. Disse funnene kan tyde på at elevenes strategier når de skal teste og/eller debugge koden, er like utviklet.

I utsagn 25 til 28, i kapittel 4.4.3 Evaluering, ser vi at læreren flere ganger spilte en rolle i elevenes arbeid med å evaluere sitt eget arbeid. Det er tilfeller hvor læreren selv tester koden for å få elevene oppmerksom på hva som skjer i deler av den, og det er tilfeller hvor læreren spesifikt ber elevene om å teste koden på egen hånd. Sammen med funnene i kapittel 4.3.1 Evaluering, kan lærerens instruksjoner i elevenes arbeid med evaluering, støtte opp mitt forslag om at elevenes strategier for å teste og debugge koden sin er lite utviklet. Dette er noe som Cui & Ng (2015) trekker frem i sin studie som en mulig faktor for at misoppfatninger oppstår. Forfatterne trekker frem at elevenes forkunnskaper i matematikk kan ha innvirkning på dette. Dette kommer av at elevenes problemløsningsstrategier i matematikk, skiller seg fra programmering i form av at de «kalkulerer» og «bestemmer», heller enn å «teste» løsningene sine. Det kan derfor være den naturlige logikken elevene har med seg fra før, som setter en stopper for hvordan elevene tester ut løsningene sine i arbeid med programmering (Cui & Ng, 2021). En annen faktor som kan bidra til å belyse kjennetegn ved elevenes misoppfatninger innenfor evaluering, er oppgavens kompleksitet og/eller kognitive krav (Qian & Lehman, 2017). Denne misoppfatningen kommer av at oppgavene i mange tilfeller blir så store at elementære programmeringsferdigheter blir glemt. Dette kan man se antydninger til i utsagn 11 og 12, da elevene i forkant av disse utsagnene hadde funnet, og tilsynelatende forstått, formelen som skulle benyttes, men likevel ikke var i stand til å finne feil i løsningen.

Det siste området som kan kategoriseres ved bruk av komponenter fra algoritmisk tenkning, er «generalisering». Å generalisere i programmering betyr at man er i stand til å se mønstre og sammenhenger, og at man evner å løse nye problemer basert på tidligere løsninger (Czismadia et al., 2015). I kapittel 4.3.3 Generalisering, ser vi i utsagn 14 og 15 at elevene har et ønske om å generalisere løsningen sin, men det kommer frem at elevene ikke forstår hvordan de skal gjøre det. Det å generalisere er en viktig del av matematikkfaget, og i disse tilfellene kan det se ut til at elevenes matematiske kunnskap knyttet til dette påvirker hvordan de evner å løse problemet. Det kan derfor se ut til at

elevenes matematiske kunnskap er noe som kjennetegner misoppfatningene elevene har innenfor generalisering. Dette støttes av Qian & Lehman´s (2017) studie, hvor det kommer frem at elevenes matematiske kunnskaper kan føre til misoppfatninger. I utsagn 16 ser vi eleven har forsøkt å generalisere, men har misoppfattet bruken av «gjenta for alltid» blokken, som i dette tilfellet fører til det jeg velger å kalle en overgeneralisering. Det kan i dette tilfellet være andre grunner enn matematiske forkunnskaper som påvirker resultatet til eleven. Qian & Lehman (2017) beskriver elevenes naturlige språk som en mulig hindring i arbeidet med matematikk i programmering, og i dette tilfellet kan det se ut til at elevens språklige forståelse av blokken «gjenta for alltid» fører til misoppfatning av hvordan h*n skal generalisere løsningen. Det er likevel ikke slik at det er gjentagende for empirien at det naturlige språket fører til misoppfatninger innenfor generalisering, og av den grunn vil jeg ikke betrakte dette punktet som et kjennetegn for de misoppfatningene som har oppstått i denne studien. Det er på en annen side interessant å få bekreftet at en faktor som spiller stor rolle innenfor syntaktisk kunnskap, påvirker elevenes misoppfatninger innenfor strategisk kunnskap.

I kapittel 4.4.5 Generalisering, ser vi at læreren i flere tilfeller har forsøkt å få elevene til å generalisere. I utsagn 30 til 32 kommer det frem at læreren på ulike måter prøver å formulere oppgavens kontekst slik at det skal bli tydeligere for elevene at de må se på det generelle, for å løse oppgaven på en hensiktsmessig måte. Det er derfor tydelig at læreren må hjelpe elevene på dette området. Cui & Ng (2021) har i resultatene av sin studie kommet frem til at åpne oppgaver kan være utfordrende for elevene i programmeringskontekst. I denne casen fikk elevene i utgangspunktet en oppgave basert på steg som de var nødt til å følge. Likevel preges oppgaven av at elevene kan utforme programmet sitt slik de ønsker, og skaper derfor rom for mange ulike løsningsforslag. Dette gjør at løsningsforslaget kan virke veldig abstrakt for elevene, og av den grunn kan de ha hatt vanskeligheter med å vite hva som egentlig skal bli resultatet til slutt. En annen faktor som også kan kjennetegne misoppfatningene elevene hadde i arbeidet med å generalisere, er elevenes strategier for å komme frem til en løsning (Qian & Lehman, 2017). Det ser ut til at elevene står fast i arbeidet med å generalisere, men har ingen gode strategier for å løse problemene underveis. Det har derfor vært et stort behov for veiledning av læreren i arbeidet med oppgaven.

5.4 Konkluderende refleksjoner

I denne studien har jeg tatt utgangspunkt i sentrale begreper fra algoritmisk tenkning, med formål om å skape et bilde av hva som kjennetegner 10. klasse elevers misoppfatninger i arbeid med matematikk i et blokkprogrammeringsmiljø. Hensikten med studiet er ikke å generalisere resultatet, men å kaste lys på kjennetegn som har bidratt til at deltakerne i akkurat denne studien har hatt misoppfatninger i arbeidet med problemet. I denne casen ser vi på syv 10. klasse elevers misoppfatninger i arbeid med matematikk i Scratch. For å kunne svare på hva som kjennetegner disse misoppfatningene, vil jeg først besvare forskningsspørsmålet i studien: «*På hvilke områder oppstår misoppfatninger i 10. klasse elevers arbeid med matematikk i et blokkprogrammeringsmiljø?*».

I kapittel 4.0 er det gjort en analyse hvor utsagn som kan tyde på misoppfatninger, er blitt organisert innenfor kategorier basert på komponenter av algoritmisk tenkning. Både i analysekapittelet, men også i diskusjonskapittelet, kommer det frem at noen områder er særlig gjentakende for misoppfatninger, mens andre ikke er det. Komponentene abstrahering og dekomponering kommer ikke tydelig frem i datamaterialet, noe som kan tilsi at det for denne casen ikke er et område misoppfatningene oppstår. Det vi derimot

kan se i denne studien, er at misoppfatningene elevene har i all hovedsak oppstår innenfor fire områder knyttet til algoritmisk tenkning. Dette er automatisering, algoritmebehandling, evaluering, og generalisering. Hver av disse områdene kan knyttes til èn av hovedkategoriene man kan organisere elevenes programmeringskunnskaper i. Misoppfatningene knyttet til automatisering går under hovedområdet «syntaktisk kunnskap». Algoritmebehandling ses på som en del av den «konseptuelle kunnskapen» til elevene, mens evaluering og generalisering knyttes opp mot «strategisk kunnskap». Med andre ord forteller disse funnene oss at misoppfatningene som har oppstått forekommer i samtlige av de tre hovedområdene, og ved å studere hver komponent kan vi se hva som kjennetegner hver enkelt av disse. Med dette vil jeg derfor besvare problemstillingen: «*Hva kjennetegner 10. klasse elevers misoppfatninger i arbeid med matematikk i et blokkprogrammeringsmiljø?*»

Innenfor syntaktisk kunnskap viser den empiriske dataen at automatisering er et område hvor elevene innehar misoppfatninger. Det som kjennetegner disse misoppfatningene er gjerne sammensatte faktorer, men i all hovedsak tyder resultatene på at elevenes erfaringer med programmeringsmiljøet Scratch er en vesentlig faktor for at det har oppstått problemer. Elevene uttrykte at de ikke forsto hvordan programmet skulle settes sammen, og empirien kan fortelle oss at elevene slet med syntaksen til programmet. I tillegg kan oppgavens kompleksitet ha bidratt til å skape misoppfatninger som i utgangspunktet ikke var der, fordi den la opp til et høyere kognitivt krav, noe som igjen skapte en stor arbeidsmengde og økt forvirring i selve automatiseringsprosessen. Et annet kjennetegn på misoppfatningene som oppsto innenfor syntaktisk kunnskap, er at elevenes naturlige språk påvirket hvordan de tolket de ferdiglagde blokkene i Scratch. Måten elevene prater på, ser ut til å påvirke hvordan de leser og forstår de allerede etablerte blokkene i språket, noe som gjorde at elevene misoppfattet syntaksen i Scratch.

Når det kommer til konseptuell kunnskap, finner vi i utgangspunktet to områder hvor misoppfatninger kan oppstå hos elevene. I abstrahering er det få klare funn fra datamaterialet som kan bidra til å svare på hvilke kjennetegn misoppfatningene på dette området har, og av den grunn er det ikke rom for å konkludere med at misoppfatninger knyttet til abstrahering er noe som kjennetegner denne casen. Det andre området innenfor konseptuell kunnskap, algoritmebehandling, var mer etablert i datamaterialet. Her finner vi flere kjennetegn som kan si noe om misoppfatningene elevene hadde underveis i arbeidet. Det som kommer tydeligst frem er at elevenes matematiske forkunnskaper ser ut til å skape misoppfatninger. Derfor anses dette som et kjennetegn på elevenes misoppfatninger i arbeid med matematikk i programmering. Et annet kjennetegn som kommer frem på dette området, er at de mentale modellene til elevene er ukomplette eller utilstrekkelige, og skaper vansker med å løse oppgaven på en hensiktsmessig måte.

Innenfor den siste hovedkategorien, strategisk kunnskap, finner vi tre områder hvor misoppfatninger kan oppstå. Dekomponering blir av samme grunn som abstrahering ikke trukket frem i konklusjonen, fordi mangel på funn i datamaterialet innenfor dette området gjør at det ikke kan anses som typisk for denne casen. De andre to komponentene, evaluering og generalisering, vil jeg derimot løfte frem. Innenfor området «evaluering» ser vi at elevene i mange tilfeller ikke evaluerer sitt eget arbeid. Det vil si at de ikke tester eller debugger programmet sitt. Dette gjelder spesielt i de situasjonene hvor det har oppstått feil i koden. I de fleste tilfellene ser vi at læreren går gjennom eller debugger koden sammen med, eller for, elevene. Det ser derfor ut til at elevenes misoppfatninger kan kjennetegnes av at de ikke har utviklet gode nok strategier i testing og debugging av kodene. I tillegg ser det ut til at elevene har blitt påvirket av kompleksiteten av oppgaven,

noe som har ført til at elevenes evne til å teste og debugge koden har blitt svekket underveis i arbeidet. Etter som dette er en del av den strategiske kunnskapen, er det grunn til å spekulere i om elevenes problematikk innenfor konseptuell kunnskap, nærmere bestemt algoritmebehandling, kan ha påvirket til dette. Innenfor generalisering er det flere kjennetegn som peker seg ut. Det virker å være en faktor at elevene har problemer med å forstå det matematiske, altså at elevene ikke vet hvordan de skal generalisere den matematiske løsningen de har kommet frem til. Dette kan bunne i elevenes forkunnskaper i matematikk, men det kan også være preget av at løsningen på oppgaven har en såpass åpen tilnærming at det som skal generaliseres blir for abstrakt. I tillegg kan det komme som et resultat av elevenes misoppfatninger innenfor algoritmebehandling, som er et område i konseptuell kunnskap. Det siste kjennetegnet på misoppfatninger elevene har hatt innenfor generalisering, er at strategiene de har benyttet har vært lite velutviklet. Mangel på gode strategier har tilsynelatende vært en gjenganger i arbeidet med oppgaven, og vises også i andre områder hvor det har oppstått misoppfatninger. Det kan være at de ukomplette mentale modellene som kommer til syne i misoppfatningene innenfor konseptuell kunnskap, har bidratt til at strategiene elevene har i arbeidet blir svekket. Det kan også være at elevenes problematikk innenfor syntaktisk kunnskap, har bidratt til å skape økt forvirring i andre deler av elevenes algoritmiske tenkning.

6 Ettertanker

Ved å studere tematikken programmering i matematikkdidaktikk over lengre tid, har jeg gjort meg ulike erfaringer og tanker. Denne studien får frem at algoritmisk tenkning i ulik grad kan være utfordrende for elevene. I mitt arbeid med denne studien fikk jeg anledning til å observere elevenes tankeprosesser og løsningsmetoder i deres arbeid med matematiske oppgaver i et blokkprogrammeringsspråk. Det er mye som tyder på at fagfornyelsens forsøk på å implementere algoritmisk tenkning i skolen, ikke er velutviklet og etablert i skolehverdagen per dags dato. Mye kan tyde på at elevenes arbeidsmetoder fortsatt baseres på arbeidsmetoder som er blitt brukt i den tidligere læreplanen, noe som heller ikke er usannsynlig med tanke på hvor liten tid programmering har vært en del av dagens læreplan. Det er store sjanser for at mer arbeid med programmering i skolen, og bedre kjennskap til programmeringsspråket som blir benyttet, vil ha positiv effekt på elevenes problemløsningsstrategier i arbeidet med ulike programmeringsmiljø.

6.1 Veien videre

En ting jeg vil ta med meg fra denne studien, er at det virker som elevenes strategier og tenkemåter har behov for å utvikles. I en lærerhverdag vil det derfor være et behov for å øve på ferdigheten å tenke algoritmisk. Det trenger ikke å være matematikk som tema, og det trenger heller ikke å være et programmeringsspråk. Det kan like enkelt være å øve på å abstrahere essensen i et problem, eller å forklare og forstå algoritmiske uttrykk. Dette kan øves på uten å nevne programmering i seg selv, og med et økt fokus på dette kan elevenes algoritmiske tankegang utvikles gradvis. Qian & Lehman (2017) foreslår å spesifikt la elevene arbeide med testing og debugging av koder som er feil, for å øve seg i å evaluere. Med andre ord kan en bruke funnene fra denne studien til å se på hva som kan være potensielle fallgruver når elevene arbeider med algoritmisk tenkning, og skape undervisningsopplegg som legger til rette for å øve på ferdighetene som trengs i de ulike områdene.

En faktor som kan spille inn på dette, er at lærerne innehar den kompetansen som er nødvendig for å drive undervisning og læring i programmering. Det kan med andre ord være en fordel at skoler og institusjoner legger til rette for at lærere og lærerstudenter får anledning til å fordype seg i dette området. På denne måten vil kunnskapsgrunnlaget som finnes i skolen bli større, og med dette kan man anta at også kvaliteten på læring og undervisning i programmering vil øke. Dette er et behov Grover & Basu (2017), påpeker i sin artikkel, hvor det kommer frem at det er nødvendig med velutviklede pedagogiske strategier og vurderinger. Skal vi tro Tabesh (2017), er algoritmisk tenkning en ferdighet som alle elever i det 21. århundre har behov for. Det vil derfor være en lang prosess for å utvikle elevenes ferdighet i algoritmisk tenkning. En kan ikke snu om etablerte mønstre og strategier på dagen, selv om fagfornyelsen tilsier at dette skal være en del av skolehverdagen. Det er derfor viktig at en arbeider videre med denne tematikken. Basert på dette håper jeg vi i snar fremtid vil få programmering inn som et fordypningstema i lærerutdanningene, og at skoler velger å satse på videreutdanning for sine ansatte innenfor dette området. Det kan også nevnes at kursing av lærere også kan være formålstjenlig for å skape en bredere kunnskap på området, og mitt håp er at det på sikt vil være obligatorisk

å benytte komponenter fra algoritmisk tenkning i undervisning, på lik linje som med andre grunnleggende ferdigheter.

6.2 Begrensninger ved studien

I denne studien har jeg tatt utgangspunkt i en gruppe på syv elever på 10. trinn, som frivillig ønsket å være med i forskningsprosjektet. Utvalget i denne studien er derfor ikke stort, og av den grunn vil det ikke gi rom for å generalisere forskningens resultater. I tillegg er utvalget basert på frivillighet, noe som betyr at deltakerne i studien har hatt et ønske om å jobbe med denne tematikken. Ved å benytte et slikt utvalg, representerer ikke denne casen et bilde av 10. klasse elever generelt, og det er derfor naturlig å anta at resultatene kunne blitt annerledes med andre deltakere.

En annen begrensning ved studien er at den er foretatt med min personlige bakgrunn, min kjennskap til tematikken, og mine generelle holdninger. Dette er faktorer som er med på å danne grunnlaget for hvordan resultatene i studien blir tolket, og det kan derfor være subjektive preg i forskningen. Det skal likevel sies at forskningen er forsøkt å gjennomføres så objektiv som mulig, men det finnes likevel alltid en form for subjektivitet bak tolkninger og spekulasjoner i forskning (Postholm & Jacobsen, 2018).

Referanser

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads, 2*(1), 48- 54

Braun, V. & Clarke, V. (2006). Using Thematic Analysis in Psychology. *Qualitative Research in Psychology. 3(2),* 77-101. https://doi.org/10.1191/1478088706qp063oa

Brennan, K. & Resnick, M. (2012). New Frameworks for Studying and Assessing the Development of Computational Thinking. *Proceedings of the 2012 Annual Meeting of the American Educational Research Association, Vol. 1,* 1-25

Bryman, A. (2016). *Social Research Methods*. (5. utg.). Oxford University Press

Bueie, H. (2019). *Programmering for matematikklærere*. (1.utg). Universitetsforlaget

Cohen, L., Lawrence, M. & Morrison, K. (2011). *Research Methods in Education*. (7. utg.). Routledge, Taylor and Francis Group

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C. & Woollard, J. (2015). *Computational thinking - A guide for teachers*. https://eprints.soton.ac.uk/424545/1/150818_Computational_Thinking_1_.pdf

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

Gjøvik, Ø., Torkildsen, H. A. (2019). Algoritmisk tenkning. *Tangenten – tidsskrift for matematikkundervisning*, 30(3). 31-37.

Grover, S., Basu, S., Bienkowski, M., Eagle, M., Diana, N., & Stamper, J. (2017). A Framework for Using Hypothesis-Driven Approaches to Support Data-Driven Learning Analytics in Measuring Computational Thinking in Block-Based Programming Environments. *ACM Transactions on Computing Education, 17(3),* 14:1-14:25. https://doi.org/10.1145/3105910

Grover, S. & Basu, S. (2017). Measuring Students Learning in Introductory Block-Based Programming: Examining Misconceptions of Loops, Variables and Boolean Logic. *SIGCSE '17: The 48th ACM Technical Symposium on Computer Science Education, Mars 8-11, 2017, Seattle, Washington* (s. 267-272). Association for Computing Machinery, New York, NY, United States

Haraldsrud, A. (2019). *Programmering og modellering – Naturvitenskapelig programmering for videregående skole*. (1.utg). Realfagsforlaget

Johannessen, A., Tufte, P. A. & Christoffersen, L. (2011). *Introduksjon til samfunnsvitenskapelig metode*. (4.utg). Abstrakt Forlag AS

Johannessen, A., Tufte, P. A. & Christoffersen, L. (2016). *Introduksjon til samfunnsvitenskapelig metode*. (5. utg.). Abstrakt Forlag AS

Kongsnes, A. L. & Wallace, A. K. (2020). *Matemagisk 9*. (1.utg). Aschehoug undervisning

Kunnskapsdepartementet. (2017). *Verdier og prinsipper for grunnopplæringen – overordnet del av læreplanverket.* Regjeringen. https://www.regjeringen.no/no/dokumenter/verdier-og-prinsipper-for-grunnopplaringen/id2570003/

Kvale, S & Birkmann, S. (2009). *Det kvalitative forskningsintervju*. 2.utg. Gyldendal

Papert, S. (1980). *MINDSTORMS: Children, Computers, and Powerful Ideas*. (2.utg) Basic Books.

Postholm, M. B. & Jacobsen, D. I. (2018). *Forskningsmetode for masterstudenter I lærerutdanning*. (1. utg.). Cappelen Damm AS

Qian, Y & Lehman, J. (2017). Students` Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Transactions on Computing Education. 18(1). 1:1-1:24*

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J. & Silverman, B. (2009). Scratch: programming for all. *Communications of the ACM*, *52*(11), 60-67.

Sevik, K. m.fl. (2016). *Programmering i skolen* (Senter for IKT i utdanningen – rapport 2016:2). https://www.udir.no/globalassets/filer/programmering_i_skolen.pdf

Skeie, J. H. (2017). *Introduksjonshefte til Scratch-programmering*. Merkur Grafisk AS

Tabesh, Y. (2017). Computational Thinkning: A 21st Century Skill. *Olympiads in Informatics, 11(2)*, 65-70

Thagaard, T. (2018). *Systematikk og innlevelse – En innføring i kvalitative metoder*. (5.utg). Fagbokforlaget.

Tjora, A. (2021). *Kvalitative forskningsmetoder i praksis*. (4. utg.). Gyldendal Norsk Forlag AS

Utdanningsdirektoratet. (2020). *Læreplan i matematikk (MAT01-05).* https://www.udir.no/lk20/mat01-05/kompetansemaal-og-vurdering/kv14?lang=nob

Wing, J. (2006). Computational Thinking. *Communications of the ACM, 49(3), 33-35*

Vedlegg

Vedlegg 1: Godkjenning fra NSD

Vedlegg 2: Samtykkeskjema

Vedlegg 1:

Meldeskjema / Masteroppgave / Eksport

Meldeskjema

Referansenummer
946794

Hvilke personopplysninger skal du behandle?

- Navn (også ved signatur/samtykke)
- Bilder eller videoopptak av personer
- Lydopptak av personer

Prosjektinformasjon

Prosjekttittel

Masteroppgave

Begrunn behovet for å behandle personopplysningene

I mitt prosjekt skal jeg analysere og se på hvordan elevers metoder og tenkemåter i et tema i matematikk kommer frem. Jeg trenger derfor å innhente opplysninger om elevene i form av hvilke metoder de bruker, hvordan de arbeider, hva de tenker rundt oppgaver de gjør, og lignende.

Ekstern finansiering

Type prosjekt
Studentprosjekt, masterstudium

Kontaktinformasjon, student
Siv Andrea Gulaker, sivagulaker@hotmail.com, tlf: 41636929

Behandlingsansvar

Behandlingsansvarlig institusjon
Norges teknisk-naturvitenskapelige universitet / Fakultet for samfunns- og utdanningsvitenskap (SU) / Institutt for lærerutdanning

Prosjektansvarlig (vitenskapelig ansatt/veileder eller stipendiat)
Hermund Andre Torkildsen, hermund.a.torkildsen@ntnu.no, tlf: 73412778

Skal behandlingsansvaret deles med andre institusjoner (felles behandlingsansvarlige)?
Nei

Utvalg 1

Beskriv utvalget

Elever fra åttende til tiende trinn

Rekruttering eller trekking av utvalget

Jeg vil ta et utvalg av elever fra skoler jeg har kontakter. Elevene vil være i grupper på 3-6 stykker. Jeg vil kontakte avdelingsleder eller rektor på den aktuelle skolen, for å få innsikt i hvilke klasser og grupperinger som kan være passende og ikke. Deretter vil jeg ta kontakt med kontaktlærer i eventuelle klasser slik at samtykkeskjema og lignende blir formidlet til foresatte.

Alder
13 - 16

Inngår det voksne (18 år +) i utvalget som ikke kan samtykke selv?
Nei

Personopplysninger for utvalg 1

- Navn (også ved signatur/samtykke)
- Bilder eller videoopptak av personer
- Lydopptak av personer

Hvordan samler du inn data fra utvalg 1?

Personlig intervju

Vedlegg

Intervjuguide.docx

Grunnlag for å behandle alminnelige kategorier av personopplysninger
Samtykke (art. 6 nr. 1 bokstav a)

Hvem samtykker for barn under 16 år?
Foreldre/foresatte

Hvem samtykker for ungdom 16 og 17 år?
Foreldre/foresatte

Informasjon for utvalg 1

Informerer du utvalget om behandlingen av opplysningene?
Ja

Hvordan?
Skriftlig informasjon (papir eller elektronisk)

Informasjonsskriv

Samtykke:infoskriv.doc

Tredjepersoner

Skal du behandle personopplysninger om tredjepersoner?
Nei

Dokumentasjon

Hvordan dokumenteres samtykkene?

- Manuelt (papir)

Hvordan kan samtykket trekkes tilbake?

Ved å sende en e-post til meg eller en av veilederne.

Hvordan kan de registrerte få innsyn, rettet eller slettet opplysninger om seg selv?

En kan få innsyn i, rettet eller slettet opplysninger ved å ta kontakt med meg på min e-postadresse, som vil bli gitt ut på informasjonsskrivet.

Totalt antall registrerte i prosjektet
1-99

Tillatelser

Skal du innhente følgende godkjenninger eller tillatelser for prosjektet?

Behandling

Hvor behandles opplysningene?

- Maskinvare tilhørende behandlingsansvarlig institusjon

Hvem behandler/har tilgang til opplysningene?

- Prosjektansvarlig
- Student (studentprosjekt)

Tilgjengeliggjøres opplysningene utenfor EU/EØS til en tredjestat eller internasjonal organisasjon?
Nei

Sikkerhet

Oppbevares personopplysningene atskilt fra øvrige data (koblingsnøkkel)?
Ja

Hvilke tekniske og fysiske tiltak sikrer personopplysningene?

- Opplysningene krypteres under lagring
- Opplysningene anonymiseres fortløpende

Varighet

Prosjektperiode
03.01.2022 - 31.05.2022

Skal data med personopplysninger oppbevares utover prosjektperioden?
Nei, data vil bli oppbevart uten personopplysninger (anonymisering)

Hvilke anonymiseringstiltak vil bli foretatt?

- Annet

NTNU vil lagre dataene kryptert. Utstyret vil lånes og transkripsjonene vil slettes etter hvert.

Vil de registrerte kunne identifiseres (direkte eller indirekte) i oppgave/avhandling/øvrige publikasjoner fra prosjektet?
Nei

Tilleggsopplysninger

Vedlegg 2: Samtykkeskjema

Vil du delta i forskningsprosjektet

Masteroppgave i Matematikkdidaktikk?

Dette er et spørsmål til deg om å delta i et forskningsprosjekt hvor formålet er å undersøke ungdomsskoleelevers oppfatning av variabel-begrepet i programmering. I dette skrivet gir vi deg informasjon om målene for prosjektet og hva deltakelse vil innebære for deg.

Formål
Dette er en avsluttende masteroppgave fra NTNU, hvor formålet med oppgaven er å undersøke i hvilken grad ungdomsskoleelever oppfatter variabel-begrepet innenfor programmering i matematikk. Programmering er blitt implementert i den nye læreplanen (LK20), men foreløpig finnes det lite relevante norske studier på området. Formålet er derfor å få en større innsikt i hvordan elever i norsk ungdomsskole arbeider med programmering, og eventuelt finne ut hvilke misforståelser som kan oppstå underveis.

Hvem er ansvarlig for forskningsprosjektet?
NTNU er ansvarlig for prosjektet.

Hvorfor får du spørsmål om å delta?
I denne oppgaven tas det utgangspunkt i elever på ungdomstrinnet, og siden du er foresatt til en elev på ungdomsskolen får du dette skrivet. I studien vil elever på ungdomsskolen bli delt inn i grupper på 3-6 stykker, hvor vi vil jobbe med programmering i matematikk. Det er gjort et tilfeldig utvalg (eventuelt: «i samråd med kontaktlærer er det gjort et utvalg ...») av elever som får være med i studien.

Hva innebærer det for ditt barn å delta?
Studiet legger opp til at mindre grupper med elever blir tatt ut av klasserommet for å være med i undervisning knyttet til programmering i matematikk. De elevene som inngår i prosjektet, vil få ulike oppgaver som skal løses. Under oppgaveløsning vil det bli tatt videoopptak av skjermen, hvor man kan se hva som blir gjort, i tillegg til at lyd også vil bli tatt opp. Underveis vil jeg samtale med elevene, og stille de spørsmål/gjøre et intervju knyttet til oppgavene. Også her vil lyden bli tatt opp. Jeg kommer til å transkribere samtalene, og de vil bli lagret kryptert hos NTNU under arbeid med prosjektet.

Det er mulighet for å se over intervjuguiden på forhånd, om det er ønskelig. Da kan du ta kontakt med meg eller veileder på e-post.

Det er frivillig å delta
Det er frivillig å delta i prosjektet. Hvis du velger å la ditt barn delta, kan du når som helst trekke samtykket tilbake uten å oppgi noen grunn. Alle dine personopplysninger vil da bli slettet. Det vil ikke ha noen negative konsekvenser for deg eller ditt barn hvis du ikke vil delta eller senere velger å trekke deg. Det vil heller ikke påvirke ditt forhold til skolen eller lærere.

Ditt barns personvern – hvordan vi oppbevarer og bruker ditt barns opplysninger
Vi vil bare bruke opplysningene som blir innhentet til formålene vi har fortalt om i dette skrivet. Vi behandler opplysningene konfidensielt og i samsvar med personvernregelverket. De som vil ha tilgang til opplysningene er meg, samt mine to veiledere.

Eneste formålet med å få fornavn til elevene er for å ha kontroll over hvilke elever som har fått samtykke fra foreldre. Etter at samtykke er hentet inn vil ikke navnet ha betydning i oppgaven, og det vil slettes.

Deltakerne i studien vil ikke kunne gjenkjennes i en eventuell publisering.

Hva skjer med opplysningene dine når vi avslutter forskningsprosjektet?
Opplysningene anonymiseres når prosjektet avsluttes/oppgaven er godkjent, noe som etter planen er i slutten av Mai/begynnelsen av Juni 2022.

Dine rettigheter
Så lenge ditt barn kan identifiseres i datamaterialet, har du rett til:

- innsyn i hvilke personopplysninger som er registrert om ditt barn, og å få utlevert en kopi av opplysningene,
- å få rettet personopplysninger om ditt barn,
- å få slettet personopplysninger om ditt barn, og
- å sende klage til Datatilsynet om behandlingen av ditt barns personopplysninger.

Hva gir oss rett til å behandle personopplysninger om deg?
Vi behandler opplysninger om ditt barn basert på ditt samtykke.

På oppdrag fra NTNU har NSD – Norsk senter for forskningsdata AS vurdert at behandlingen av personopplysninger i dette prosjektet er i samsvar med personvernregelverket.

Hvor kan jeg finne ut mer?
Hvis du har spørsmål til studien, eller ønsker å benytte deg av dine rettigheter, ta kontakt med:

- *NTNU* ved *Siv Andrea Gulaker, sivag@stud.ntnu.no, eller Hermund Andre Torkildsen, hermund.a.torkildsen@ntnu.no eller Marit Buseth Langfeldt, marit.b.langfeldt@ntnu.no*
- Vårt personvernombud: *Thomas Helgesen, thomas.helgesen@ntnu.no*

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

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

Med vennlig hilsen

Hermund Andre Torkildsen
Marit Buseth Langfeldt
(Forsker/veileder)

Siv Andrea Gulaker

Samtykkeerklæring

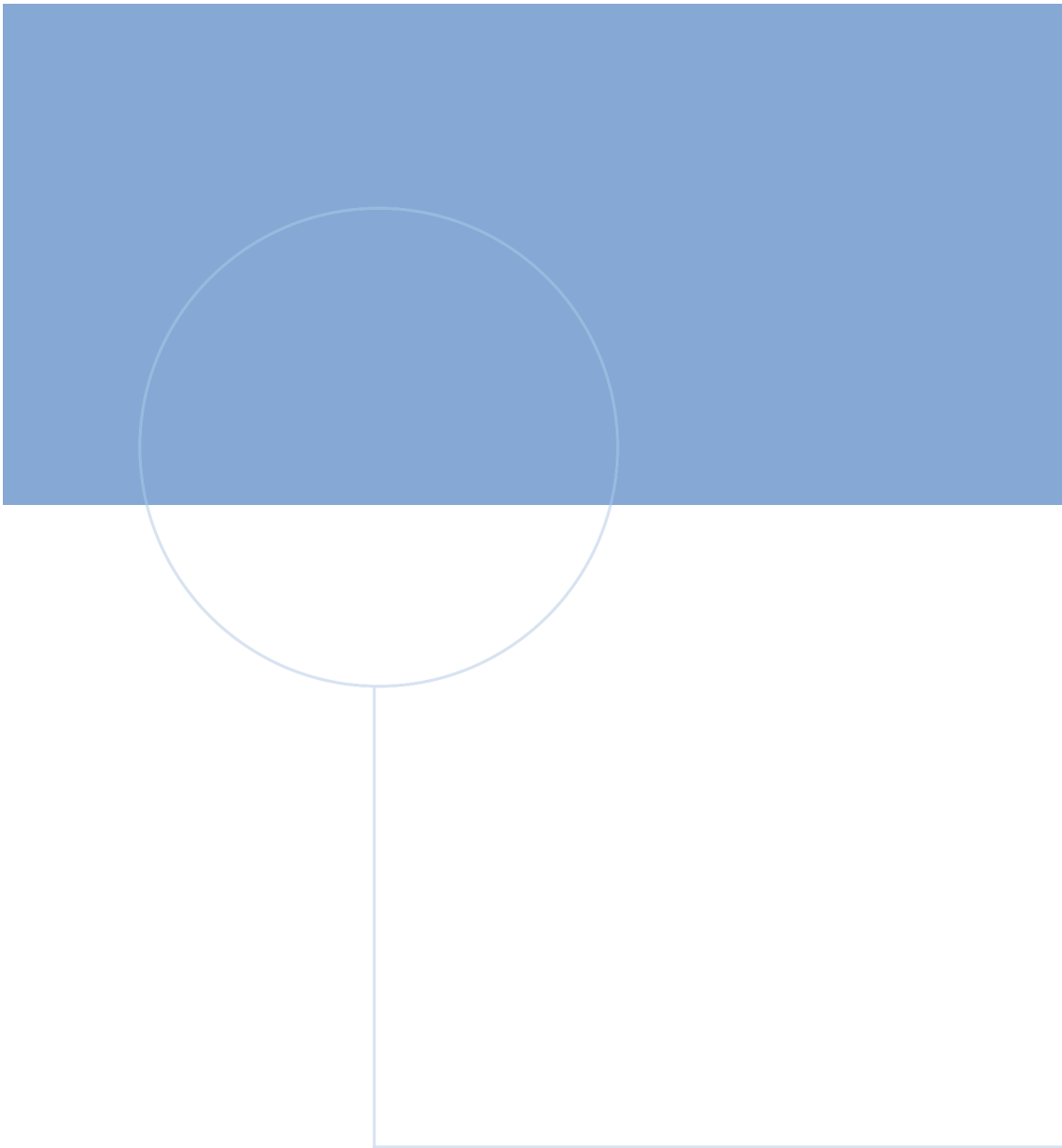
Jeg har mottatt og forstått informasjon om prosjektet *Masteroppgave i Matematikkdidaktikk* og har fått anledning til å stille spørsmål. Jeg samtykker til:

- ☐ å la mitt barn delta i *intervju*
- ☐ å la mitt barn delta i *spørreskjema/oppgaver – hvis aktuelt*

Jeg samtykker til at opplysninger om mitt barn behandles frem til prosjektet er avsluttet

---

(Signert av foresatt til prosjektdeltaker, dato, fornavn på prosjektdeltaker)



NTNU

Kunnskap for en bedre verden