UiT Norges arktiske universitet

Institutt for lærerutdanning og pedagogikk

Algoritmisk tenkning i matematikk

En kvalitativ casestudie om hva som kjennetegner elevers algoritmiske tenkning.

Elisabeth Dahl Olafsen

Masteroppgave i lærerutdanning, 5.- 10. trinn, mai 2021

LRU-3903 Matematikkdidaktikk

Forord

En masteroppgave i matematikkdidaktikk markerer slutten på grunnskolelærerutdanningen for 5.-10.trinn ved universitetet i Tromsø. Samtidig markerer den starten på læreryrket. Jeg ser fram til å praktisere alt jeg har lært i årene som kommer.

Takk til veileder Jan Nyquist Roksvold for gode diskusjoner og tilbakemeldinger underveis. Jeg setter pris på at du har vist interesse og nysgjerrighet for mitt prosjekt. Jeg vil også rette en takk til elevene som deltok på min studie, uten dere hadde ikke denne oppgaven blitt til. Til slutt vil jeg takke mine kollegaer på «Den digitale Framtidssmien» ved ILP – institutt for lærerutdanning og pedagogikk. I fire år har jeg som studentansatt fått være med på et spennende samarbeid om hvordan man kan utvikle den profesjonsfaglige digitale kompetansen i lærerutdanningen og ute i praksisfeltet. Våre diskusjoner om hvordan teknologi kan implementeres i skolen og i de enkelte fag har vært til stor inspirasjon i arbeidet med å skrive denne masteroppgaven.

Tromsø, mai 2021

Elisabeth Dahl Olafsen

Sammendrag

Fra høsten 2020 ble programmering og algoritmisk tenkning tatt inn i læreplanen i matematikk. Algoritmisk tenkning blir løftet fram som en viktig ferdighet for å lykkes med programmering, men blir også sett på som en viktig problemløsningsmetode i matematikk.

Dette mastergradsprosjektet undersøker hva som kjennetegner elevers algoritmiske tenkning når arbeider med en geometrioppgave med blokkbasert programmering. Elevene programmerer ikke selv, men får utdelt ferdiglagde algoritmer som de skal tolke og analysere. Studien er et kvalitativt casestudie hvor 10 elever på 6.trinn blir filmet og observert mens de arbeider i par. Elevenes algoritmiske tenkning blir analysert induktivt og tematisk ut fra Bocconi et al. (2016) sine tre kjennetegn på algoritmisk tenkning: a*bstraksjon, generalisering* og *algoritmebehandling* (Gjøvik og Torkildsen, 2019 sine norske oversettelser).

Studiens funn viser tre typer algoritmiske tenkere: «*Den instrumentelle algoritmiske tenkeren», «Den prosedurale algoritmiske tenkeren»* og «*Den konseptuelle algoritmiske tenkeren»*. Bakgrunnen for disse er hvordan kjennetegnene abstraksjon, generalisering og algoritmebehandling kommer til uttrykk i elevenes løsningsforslag. For abstraksjon er det utviklet to kategorier med to nivåer: Nivå 1) Klarer å bruke funksjonsblokkene, og Nivå 0) Klarer ikke å bruke funksjonsblokkene. For generalisering er det utviklet to kategorier med to nivåer: Nivå 1) Oppdager mønstre i funksjonsblokkene, og Nivå 0) Oppdager ingen mønstre i funksjonsblokkene. For algoritmebehandling er det utviklet tre kategorier med tre nivåer: Nivå 2) Følger algoritmene korrekt med vet ikke svaret, Nivå 1) Bruker algoritmene som «støtte» til tidligere kunnskap, og Nivå 0) Klarer ikke å følge algoritmene. Den instrumentelle algoritmiske tenkeren viser nivå 0 på abstraksjon, nivå 0 på generalisering og nivå 0 på algoritmebehandling. Den prosedurale algoritmiske tenkeren viser nivå 1 på abstraksjon, nivå 0 på generalisering og nivå 2 på algoritmebehandling. Den konseptuelle algoritmiske tenkeren viser nivå 1 på abstraksjon, nivå 1 på generalisering, og nivå 1 på algoritmebehandling.

Studien konkluderer med at den prosedurale algoritmiske tenkeren og den konseptuelle algoritmiske tenkeren til sammen viser flest kjennetegn på algoritmisk tenkning.

Innholdsfortegnelse

1 Innledning 1
1.1 Bakgrunn for mastergradsprosjektet 1
1.2 Valg av programmeringsspråk 2
1.3 Oppgavens problemstilling 4
1.4 Masteroppgavens oppbygning 5
2 Teoretisk grunnlag 7
2.1 Matematikk som ulike tenkevaner 7
2.2 Del 1: Hva er algoritmisk tenkning? 9
2.2.1 Hva er algoritmer i matematikk? 9
2.2.2 Hva er algoritmer i programmering? 11
2.2.3 Hva er «tenkning» i matematikk? 12
2.2.4 Hva er «tenkning» i programmering? 15
2.3 Del 2: Hvordan brukes begrepet i internasjonal og norsk forskningslitteratur? 16
2.3.1 I internasjonal forskningslitteratur 16
2.3.2 I norsk forskningslitteratur 18
2.4 Del 3: Hva kjennetegner algoritmisk tenkning? 21
2.4.1 Bocconi et al. (2016): Et rammeverk for *computational thinking* 22
3 Metode 27
3.1 Studiens kunnskapssyn 27
3.2 Valg av metode 27
3.2.1 Instrumentell casestudie 28
3.2.2 Videoopptak som metode for datainnsamling 28
3.2.3 Observasjon som metode for datainnsamling 29
3.3 Utvelgelse av informanter 30
3.4 Forberedelse og gjennomføring av studien 30

3.4.1 Utforming av oppgaven 30
3.4.2 Begrunnelse for valg av oppgave 31
3.5 Analysemetode 33
3.5.1 Induktiv og tematisk analyse 34
3.6 Kvalitet i studien 38
3.6.1 Validitet 39
3.6.2 Reliabilitet 40
3.7 Etiske retningslinjer 42
4 Funn og analyse av funn 45
4.1 Tre typer algoritmiske tenkere 45
4.2 Abstraksjon 48
4.3 Generalisering 51
4.4 Algoritmebehandling 56
5 Diskusjon 65
5.1 Den instrumentelle algoritmiske tenkeren 65
5.2 Den prosedurale algoritmiske tenkeren 67
5.3 Den konseptuelle algoritmiske tenkeren 69
5.4 Hva kjennetegner algoritmisk tenkning? 70
6 Avslutning 73
6.1 Hva har jeg funnet ut? 73
6.2 Teoretiske og didaktiske implikasjoner 74
6.3 Hvordan forske videre på dette? 75
Vedlegg 1: Oppgaven 81
Vedlegg 2: Informasjonsskriv og samtykkeskjema 82
Vedlegg 3: Godkjennelse fra NSD 85

Tabelliste

Tabell 1: Den egyptiske multiplikasjonsalgoritmen. Hentet fra Bueie (2019, s. 29) 10
Tabell 2: Studiens funn: Tre typer algoritmiske tenkere. 46
Tabell 3:Tabellen viser at den prosedurale algoritmiske tenkeren (PRO.AT) og den konseptuelle algoritmiske tenkeren (KON.AT til sammen viser flest kjennetegn på algoritmisk tenkning. 71

Figurliste

Figur 1: Tomt Scratch-vindu. Hentet fra Scratch.mit.edu 3
Figur 2: Blokkbasert og grafisk framstilling av en stjerne. Laget i Scratch. 11
Figur 3 Bocconi et al. (2016) sine seks kjennetegn på computational thinking (figur t.v). Gjøvik og Torkildsen (2019) sine norske oversettelser (figur t.h). 22
Figur 4 : Algoritmen til venstre er en abstraksjon av en femkantet stjerne. Laget i Scratch. .. 24
Figur 5: Oppgaven som elevene fikk. Algoritmene er laget i scratch.mit.edu 31
Figur 6: Fasit på oppgaven. Pilene viser hvilken algoritme som hører til den rette firkanten. 32
Figur 7: Illustrasjon av et rektangel tegnet av en elev. 35
Figur 8: Koding i Nvivo. 37
Figur 9: Elevenes oppgaveark. Bildet viser hvordan de har forsøkt å representere gå-blokken som prikker og hvordan de har rotert i planet. 50
Figur 10: Rombealgoritmen (t.v), kvadratalgoritmen (t.h). 52
Figur 11: Rektangelalgoritmen 53
Figur 12: Rombealgoritmen (t.v), Kvadratalgoritmen (t.h). 54
Figur 13: Parallellogramalgoritmen 57
Figur 14: En elev følger instruksjoner (figur t.v), figuren man ville fått i Scratch (figur t.h).. 58
Figur 15: Rombealgoritmen 58
Figur 16: En elev følger instruksjoner (figur t.v), figuren man ville fått i Scratch (figur t.h).. 59
Figur 17: En elev tegner opp kvadratalgoritmen. 60
Figur 18: En elev tegner opp rektangelalgoritmen. 60
Figur 19: Illustrasjon av hvordan en elev prøver å følge parallellogramalgoritmen. 62

Figur 20 Bocconi et al. (2016) sine seks kjennetegn på computational thinking (t.v). Gjøvik og Torkildsen sine norske oversettelser (t.h), hentet Gjøvik, Ø. & Torkildsen, H. A. (2019). Algoritmisk tenkning. *Tangenten–tidsskrift for matematikkundervisning, 30*(3), 31-37..... 80

Kapittel 1

1 Innledning

1.1 Bakgrunn for mastergradsprosjektet

Opp gjennom tidene har det vært diskutert hvilken plass programmering skal ha i skolen og om det i det hele tatt skal være en del av elevenes opplæring (NOU 2013:2, s. 101). Bueie (2019, s. 19) skriver at vi i dag kan legge denne diskusjonen bak oss da vi i fagfornyelsen for 2020 ser at matematikkfaget får en sentral rolle i opplæring av elever i programmering. Dette stiller krav til dagens og fremtidens matematikklærere: ikke bare må de ha kunnskaper og ferdigheter i programmering, men de må også klare å integrere programmering på matematikkfagets premisser.

Algoritmisk tenkning blir ofte nevnt i sammenheng med programmering og er i fagfornyelsen 2020 løftet fram som en viktig problemløsningsmetode i matematikk (Utdanningsdirektoratet, 2020c). Ifølge Gjøvik og Torkildsen (2019, s. 31) er det ikke overenstemmelse mellom hvordan dette begrepet skal defineres eller forstås i internasjonal og norsk forskningslitteratur. I internasjonal forskningslitteratur brukes begrepet *computational thinking* hvor *algorithmic thinking* blir sett på som en viktig del av computational thinking (Bocconi et al., 2016, s. 16; Wing, 2006, s. 33). Wing (2006, s. 33) skriver at algorithmic thinking er det samme som computational thinking, men at algorithmic thinking også er en ferdig som kreves for å utvikle computational thinking. Videre skriver Wing at computational thinking både er en tankeprosess og en problemløsningsmetode som kan utvikles med og uten teknologi. I Norge har vi oversatt computational thinking direkte til algoritmisk tenkning, noe som medfører av vi mangler et tilsvarende norsk begrep for det som på engelsk kalles for *algorithmic thinking* (Gjøvik & Torkildsen, 2019).

Fra et matematikkperspektiv kan algoritmisk tenkning fremstå som et gammelt begrep i en ny kontekst da algoritmer har eksistert lenge før datamaskinen ble oppfunnet (Bueie, 2019, s. 28). Algoritmisk tenkning kan også som Gjøvik og Torkildsen (2019, s. 31) skriver: «høres ut som et gufs fra fortiden, fra den tiden da skolematematikken hadde mer fokus på algoritmiske løsninger av matematikkoppgaver». Stenseth, Kaufmann og Forsstöm (2019, s. 8) skriver at

begrepet i sin generelle form også kan forstås utenfor en matematikkontekst, da både strikkeoppskrifter og matoppskrifter kan forstås som algoritmer.

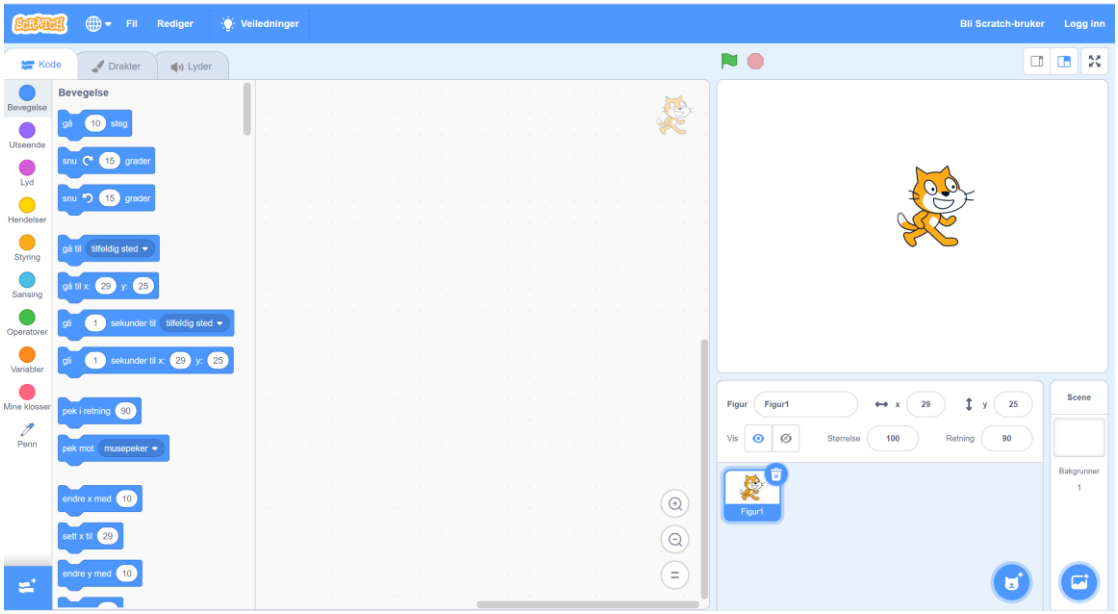
Som man ser finnes det flere måter å forstå begrepet algoritmisk tenkning på. Dersom algoritmisk tenkning skal være et fruktbart begrep å bruke i matematikkundervisningen, må man rydde opp i begrepet slik at det ikke blir en kilde til mange misforståelser. Motivasjonen bak å skrive denne masteroppgaven har derfor vært å undersøke dette begrepet nærmere for å kunne si noe mer om hva som kjennetegner elevers algoritmiske tenkning. Spørsmål jeg har stilt meg underveis i denne prosessen har vært: Hva har algoritmisk tenkning med matematikkfaget å gjøre? Er algoritmisk tenkning noe man gjør når man programmerer, eller kan det være uavhengig av at man programmerer? Er algoritmisk tenkning en egen type problemløsningsmetode? Hva skiller den i så fall fra andre problemløsningsmetoder i matematikk?

1.2 Valg av programmeringsspråk

Ifølge Gjøvik og Torkildsen (2019, s. 34) kommer det stadig nye programmeringsverktøy og programmeringsspråk som kan brukes i undervisningssammenheng. Målet med å innføre programmering i matematikkfaget bør ikke være at elever skal lære seg alle de ulike verktøyene og språkene som finnes. Målet bør være å lære matematikk *gjennom* å programmere. Dermed må inngangsterskelen til å programmere være lav slik at teknologien ikke kommer i veien for læringen. Både Gjøvik og Torkildsen (2019) og Stenseth et al. (2019, s. 7) trekker fram algoritmisk tenkning som et viktig begrep i denne sammenhengen. Grunnen til dette er at mye av tankegangen som ligger bak programmering, kan knyttes til algoritmisk tenkning. Dermed er algoritmisk tenkning igjen en viktig ferdighet for å mestre programmeringsfaget. Et spørsmål til dette er hvordan man som matematikklærere skal vite hva det vil si å tenke algoritmisk? Hvilke kjennetegn skal man se etter hos sine elever?

For å finne svar på dette har jeg i denne masteroppgaven valgt å studere elevers algoritmiske tenkning når de arbeider med geometri i blokkbasert programmering. Oppgaven som elevene skal arbeide med er uten bruk av digitale verktøy, men er utviklet i det blokkbaserte programmeringsspråket Scratch. Scratch er ifølge Haraldsrud, Sveinsson og Løvold (2020, s. 87) et av de mest utbredte blokkprogrammeringsverktøyene i dag. *Lær Kidsa Koding* – en norsk organisasjon som jobber med implementering av programmering i skolen, har basert

mange av sine kurs og undervisningsopplegg på dette språket (Haraldsrud et al., 2020, s. 88). En av grunnene til at Scratch har blitt så populært, er at det gir en god introduksjon til de viktigste programmeringskonseptene uten at elever må lære seg et tyngre programmeringsspråk. Når man åpner Scratch for første gang dukker dette bildet opp:



Figur 1: Tomt Scratch-vindu. Hentet fra Scratch.mit.edu

På venstre side kan man velge mellom 9 ulike kategorier som er merket etter farger. I oppgaven som elevene får blir det bare tatt utgangspunkt i de blå funksjonsblokkene som står for konseptet *bevegelse.* I høyre hjørne finner man katten Scratch som kan utføre handlinger gjennom programmet vi skriver. Som navnet tilsier, handler blokkprogrammering om at man setter sammen ulike blokker med ulike funksjoner til et ønsket program. Deretter kjører man programmet og resultatet dukker opp i vinduet der Scratch befinner seg. Scratch er som man kan se, et veldig visuelt språk som gjør det mulig å utforske matematiske konsepter på nye måter (Haraldsrud et al., 2020, s. 97).

Jeg ønsker ikke å gå inn i debatten om hvorvidt man burde bruke tekstbaserte eller blokkbaserte programmeringsspråk i matematikkundervisningen. Men som Haraldsrud et al. (2020, s. 13-14) skriver, kan blokkbaserte programmeringsspråk være en fin introduksjon til å forstå programmering, men at elever på et høyere alderstrinn bør lære seg tekstbasert programmering da dette gir det beste grunnlaget for å drive på med programmering senere.

1.3 Oppgavens problemstilling

På bakgrunn av dette har jeg utformet følgende problemstilling for denne masteroppgaven:

Hva kjennetegner elevers algoritmiske tenkning når de arbeider med geometri i blokkbasert programmering?

Jeg vil understreke at det først og fremst er elevenes algoritmiske tenkning som undersøkes i denne oppgaven. Dermed vil man se at geometrifaget er nedtonet i hele oppgaven.

1.4 Masteroppgavens oppbygning

Kapittel 2: Teoretisk grunnlag

Det teoretiske grunnlaget for oppgaven er delt inn i tre deler. Del 1: Hva er algoritmisk tenkning? Del 2: Hvordan brukes begrepet i internasjonal og norsk forskningslitteratur? Del 3: Hva kjennetegner elevers algoritmiske tenkning? I del 3 presenteres Bocconi et al. (2016) sitt rammeverk.

Kapittel 3: Metode

I metodekapittelet presenteres 1) Studiens kunnskapssyn, 2) Valg av metode og forskningsdesign, 3) Utvelgelse av informanter, 4) Forberedelse og gjennomføring av studien, 5) Analysemetode og 6) Kvalitet i studien.

Kapittel 4: Funn og analyse av funn

Her presenteres studiens funn og hvordan elevene gikk fram for å løse oppgaven.

Kapittel 5: Diskusjon.

Mine funn diskuteres opp mot oppgavens teoretiske grunnlag og oppgavens problemstilling.

Kapittel 6: Avslutning.

Her svarer jeg på hva jeg har funnet ut og hvilke teoretiske og didaktiske implikasjoner studiens funn kan ha. Jeg skriver også hvordan jeg ville forsket på dette videre.

Kapittel 2

2 Teoretisk grunnlag

Problemstillingen for denne masteroppgaven er: *Hva kjennetegner elevers algoritmiske tenkning når de arbeider med geometri i blokkprogrammering?* For å kunne svare på dette spørsmålet, er del 1 av teorikapittelet en begrepsavklaring på hva algoritmisk tenkning er. I del 2 ser jeg på hvordan begrepet brukes i internasjonal og norsk forskningslitteratur. I del tre presenteres Bocconi et al. (2016) sitt rammeverk som også er rammeverket for studien min. Før jeg går inn på del 1 av teorikapittelet, ønsker jeg å si noen ord om hvordan matematikk er definert i denne masteroppgaven.

2.1 Matematikk som ulike tenkevaner

I høst tok jeg fordypning i matematikkdidaktikk. Et stadig tilbakevennende spørsmål har vært: Hva er matematikk? Dette spørsmålet har vist seg å være vanskelig å besvare. Ifølge Davis, Hersh og Marchisotto (2011, s. 8) vil synet på matematikk endre seg som følge av at samfunnet endrer seg. Dermed vil svaret på dette spørsmålet variere ut ifra *hvem* man spør og i hvilken sammenheng man spør. Siden denne masteroppgaven handler om elevers tenkning, har jeg blitt inspirert av hvordan Cuoco, Goldenberg og Mark (1996) i *Habits of Mind: An Organizing Principle for Mathematics Curricula,* skriver om ulike tenkevaner i matematikk som vil være viktig for elever å utvikle. De skriver:

> If we really want to empower our students for life after school, we need to prepare them to be able to use, understand, control, modify, and make decisions about a class of technology that does not yet exist. That means we have to help them develop genuinely mathematical ways of thinking. […] The thought processes, the ways of looking at things, and the habits of mind used by mathematicians, computer scientists, and scientists will be mirrored in systems that will influence almost every aspect of our daily lives (Cuoco et al., 1996, s. 401).

Couco et al. skriver at barn og unge i skolen må rustes for å møte et samfunn som er i stadig endring. Dette innebærer at elever må kunne bruke, forstå, kontrollere, modifisere og gjøre egne beslutninger om hvordan teknologi skal brukes på en god måte. Elever må utvikle det Cucuo et al. kaller for *habits of mind* eller *ways of thinking*, som på norsk kan oversettes til:

ulike tenkevaner eller *måter å tenke på.* Elever må lære seg å tenke som matematikere, informatikere og forskere slik at de kan være med på å løse framtidige utfordringer.

Denne tilnærmingen til matematikk finner vi også i Knuth (1985) sin artikkel: *Algorithmic thinking and mathematical thinking.* Knuth er både matematiker og informatiker. Knuth skriver at for han er informatikk «studiet av algoritmer» og han har tenkt lite over sammenhengen mellom algoritmer i programmering og algoritmer i matematikk. Han stiller følgende spørsmål:

> What is the relation of algorithms to modern mathematics? Is there an essential difference between an algorithmic viewpoint and the traditional mathematical world-view? Do most mathematicians have an essentially different thinking process from that of most computer scientist? (Knuth, 1985, s. 171).

Det Knuth lurer på er om matematikere tenker på en helt annerledes måte enn det informatikere gjør. Han spør hva forskjellen er på det å bruke algoritmer i matematikk og det å bruke algoritmer i programmering. For å finne et svar på disse spørsmålene gjør Knuth følgende: han finner fram 9 bøker som alle inneholder ulike matematiske problem. Han slår tilfeldig opp på side 100 i alle de 9 bøkene og forsøker deretter å programmere en datamaskin til å løse disse 9 problemene for han. Gjennom dette «eksperimentet» kommer Knuth med følgende konklusjon: det er ingenting som heter matematisk tenkning. Man kan ikke forstå matematisk tenkning som et isolert konsept. Matematikere bruker det Knuth kaller for *modes of thought.* Dette er likt det Cuoco et al. (1996) kaller for *habits of mind*. Det Knuth mener med *modes of thought* er at når han arbeider som informatiker tenker han som en informatiker. Da er informatikk «studiet av algoritmer». Når Knuth arbeider som matematiker tenker han som en matematiker. Da er matematikk «studiet av algoritmer». Knuth skriver at med mindre man begynner å reflektere over sammenhengen mellom disse to måtene å tenke på, så er det ikke sikkert man er klar over at det eksisterer noe sammenheng.

I fagfornyelsen 2020 legges det også vekt på at matematikk handler om ulike måter å tenke på. I en pressemelding fra regjeringen står det følgende: «Elevene skal jobbe mer med metoder og tenkemåter slik at de får en større forståelse for faget» (Meld. St.20 (2018), s. 4). Dette kommer til uttrykk gjennom fem kjerneelement som har blitt en del av læreplanen i matematikk og som skal bidra til at man ivaretar det viktigste i faget. Disse er: *utforsking og problemløsning, modellering og anvendelser, resonnering og argumentasjon, representasjon*

og *kommunikasjon* og *abstraksjon og generalisering* (Utdanningsdirektoratet, 2020c). Videre står det at programmering og algoritmisk tankegang blir en del av matematikkfaget.

Denne oppgaven har den samme tilnærming til matematikk som vi finner hos Knuth (1985), Cuocu et al. (1996) og fagfornyelsen 2020, nemlig at matematikk handler om *ulike tenkevaner*.

2.2 Del 1: Hva er algoritmisk tenkning?

For å kunne svare på hva som kjennetegner elevers algoritmiske tenkning, handler del 1 av teorikapittelet om hva algoritmisk tenkning er. Algoritmisk tenkning er et sammensatt begrep. Det består av adjektivet *algoritmisk* som er avledet av substantivet *algoritme*, og substantivet *tenkning* som er avledet av verbet *tenke* (NAOB). I dette ligger det implisitt at vi må forstå hva en algoritme er og hva å tenke er. For å kunne gjøre dette på en oversiktlig måte vil jeg skille mellom disse to begrepene i en matematikkontekst og i en programmeringskontekst.

2.2.1 Hva er algoritmer i matematikk?

Et første steg mot å besvare dette spørsmålet kan være ved å se på ordets opprinnelse. Ordet *algoritme* stammer ifra den arabiske matematikeren og astronomen Mohamed al Khwarizmi født ca. år 820. På denne tiden ble begrepet brukt som en betegnelse for å regne med det arabiske tallsystemet. En *algoritmiker* var dermed en person som regnet med arabiske tall i det arabiske tallsystemet (Grønmo & Hovde, 2020).

Vi finner spor av at algoritmer har blitt brukt i Rhind-papyrusen omkring år 1550 f.v.t. Rhind-papyrusen er en papyrusrull som ble utviklet under oldtidens Egypt i Mesopotamia og er en av de viktigste kildene vi har om oldtidens egyptiske matematikk. Rhind-papyrusen inneholder 85 matematiske problemer som kan løses enten ved å bruke multiplikasjon eller divisjon som regnemetode (Burton, 1985, s. 36). Et eksempel på en algoritme fra Rhind-papyrusen, er den egyptiske multiplikasjonsalgoritmen.

1	8
2	16
4	32
8	64
16	128

Tabell 1: Den egyptiske multiplikasjonsalgoritmen. Hentet fra Bueie (2019, s. 29)

Si at du ønsker å finne ut hva 18 x 8 blir. Egypterne løste dette ved å bruke en doblingsalgoritme (Bueie, 2019, s. 28). Man velger et av de to tallene som man ønsker å multiplisere, for eksempel tallet 8 og lager doblinger av dette tallet (høyre kolonne). Deretter velger man ut de tallene i venstre kolonne som til sammen gir summen 18. I mitt tilfelle er det tallet 16 + 2. Dermed må 18 x 8 = 128 + 16 = 144. Et annet eksempel på en gammel algoritme, er Euklids algoritme for å finne største felles divisor for to heltall. Denne algoritmen ble beskrevet i Euklids verk *elementer* rundt 300 f.v.t i den delen som omhandlet tallteori (Burton, 1985, s. 173). Å skulle gå nærmere inn på hvordan Euklids algoritme fungerer, ligger utenfor rammene for denne oppgaven, men det viktigste er at Euklids algoritme gjorde det mulig å finne den største felles divisoren for store tall på en effektiv måte.

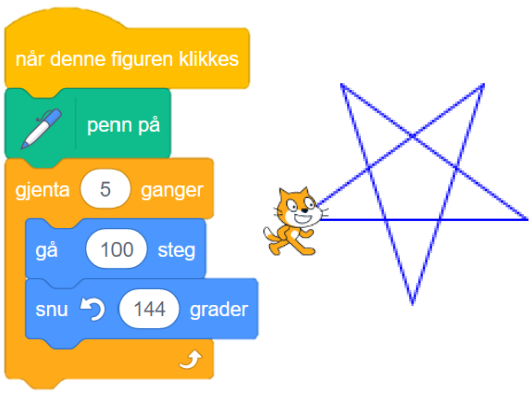
I matematikk finnes det mange algoritmer. En algoritme i matematikk er en ordnet liste med instrukser som viser hvordan du må gå fram for å løse et matematisk problem. Videre kjennetegnes algoritmer av at de kan løse en hel klasse av like matematiske problemer (Brousseau, 2006, s. 129). Dette gjør også at algoritmer er effektive. Fra vår egen skolegang er kanskje multiplikasjonsalgoritmen og divisjonsalgoritmen to nærliggende eksempler på dette. Dersom du lærer deg stegene i multiplikasjonsalgoritmen eller divisjonsalgoritmen, vil det være mulig å løse et sett med liknende matematiske problem.

2.2.2 Hva er algoritmer i programmering?

Algoritmer har eksistert lenge før datamaskinen ble oppfunnet, men de ble spesielt betydningsfulle ved inntoget av den moderne datamaskinen (Bueie, 2019, s. 28). Nå kunne man implementere algoritmer til en datamaskin og få datamaskinen til å gjøre en rekke beregninger for deg.

Stenseth et al. (2019, s. 8) skriver at en algoritme i sin generelle form kan forstås utenfor en matematikkontekst. For eksempel kan en strikkeoppskrift, en bruksanvisning eller en matoppskrift forstås som algoritmer. En måte å skille algoritmer i matematikk på fra algoritmer i programmering, er ved å se på hvordan en algoritme er skrevet. En algoritme i programmering er bygd opp av det som kalles for koder (Haraldsrud et al., 2020, s. 15). En kode kan skrives på ulike måter, derfor skiller man gjerne mellom ulike programmeringsspråk og mellom tekstbasert og blokkbasert programmering. En kode kan bestå av tall, bokstaver, symboler og andre tegn. Alt avhenger av hvilket programmeringsspråk man bruker (Istad & Kristoffersen, 2019, s. 9-15). Koding er mer snevert enn programmering fordi det innebærer selve aktiviteten med å reformulere ideer og konsepter inn i et bestemt programmeringsspråk (Gjøvik & Torkildsen, 2019, s. 34).

Noe som jeg ofte hører er at begrepene *kode*, *algoritme*, *program* og *programmering* blir brukt om hverandre. Dette vil jeg forsøke å avklare med et eksempel fra blokkbasert programmering og med begrepsforklaringene til vitensentrene (vitensenter, 2018):



Figur 2: Blokkbasert og grafisk framstilling av en stjerne. Laget i Scratch.

Figur 2 består av flere *koder* som er skrevet i det blokkbaserte programmeringsspråket Scratch. Til sammen utgjør disse kodene en *algoritme*. En algoritme er en ordnet liste med instrukser som viser hvordan du må gå fram for å løse et bestemt problem (Bueie, 2019, s. 28). Samtidig er også dette et *program* fordi den forteller en datamaskin hva den må gjøre for at man skal få et ønsket resultat. Når man kjører dette programmet blir disse kodene bearbeidet og omgjort til en handling. I dette tilfellet får vi en grafisk framstilling av en stjerne (se figur over). *Programmering* viser til hele prosessen fra å utvikle og teste koder, dele de opp i mindre biter, analysere dem og til slutt implementere dem til en datamaskin på et språk som datamaskinen forstår (Stenseth et al., 2019, s. 7). Ordet programmering forekommer dermed i mange forskjellige sammenhenger i dag fordi vi omgir oss av mange automatiserte og programmerte gjenstander (Bueie, 2019 s. 22).

Det kan være forvirrende at begrepet algoritmisk tenkning nevnes i sammenheng med både programmeringsfaget og matematikkfaget. Er det snakk om at elever skal utvikle forståelse for standardalgoritmer i matematikk, eller er det snakk om at de skal programmere matematiske algoritmer på en datamaskin? Algoritmer i matematikk og algoritmer i programmering brukes til ulike formål, men i begge tilfeller er de et sett med instrukser som viser hvordan man må gå fram for å løse et problem. En forskjell kan være at algoritmer i matematikk brukes for å løse bestemte matematiske problemer, mens algoritmer i programmering brukes til å løse en rekke ulike problemer. Dermed kan det tenkes at ordet algoritme brukes mer generelt i programmering enn i matematikk.

2.2.3 Hva er «tenkning» i matematikk?

Tenkning er avledet av verbet *tenke* (NAOB). *«Å tenke»* er knyttet til prosessen med å bearbeide mentale sanseinntrykk (Svartdal, 2018). Dette kan være alt fra ting man husker, forestillinger, følelser, bilder og verbale symboler. En utfordring med å bruke ordet *tenkning* er at det er vanskelig å få tilgang til hvordan elever tenker siden dette foregår på et indre plan. Hvordan skal matematikklærere vite at elever tenker algoritmisk? Gjøvik & Torkildsen (2019, s. 31) skriver at det er begrepet *tankegang* fremfor *tenkning* som ser ut til å vokse fram i Norge i tilknytning til algoritmisk tenkning. De referer til en litteraturanalyse som er gjort av Bocconi, Chioccariello og Earp (2018):

> In Norway, algoritmisk tankegang (sic) (EN: algorithmic thinking) emerges as the most widely used umbrella term […] (Bocconi et al., 2018, s. 8).

Gjøvik og Torkildsen skriver at en av grunnene til at man i Norge har valgt *tankegang* framfor *tenkning*, kan være at *tenkning* oppfattes som noe mer generelt enn *tankegang.* De mener at det kan være mer fruktbart å bruke ordet *tenkning* i stedet for *tankegang* fordi man i matematikkfaget gjerne er opptatt av generelle måter å tenke og jobbe på. Det er uklart hva Gjøvik og Torkildsen mener med akkurat dette, men det kan tenkes at *tankegang* blir for snevert for å fange opp hele essensen bak det *å tenk*e algoritmisk.

Denne oppgaven tar utgangspunkt i at ordet *tenkning* er for abstrakt til at man kan måle dette direkte hos elever. Dermed vil tenkning forstås i lys av matematisk forståelse gjennom å se på Skemp (1976) sine begreper om *instrumentell* og *relasjonell* forståelse i matematikk og Hiebert og Lefevre (1986) sine begreper om *prosedural* og *konseptuell* forståelse i matematikk.

Richard Skemp – Instrumentell og relasjonell forståelse i matematikk

Richard Skemp (1976) er kjent for sine to måter å skille matematisk forståelse på. Disse har han kalt for *instrumentell* og *relasjonell* forståelse i matematikk. Instrumentell forståelse i matematikk er ifølge Skemp «rules without reason» som kan oversettes til «regler uten begrunnelse». Elever som har en instrumentell forståelse i matematikk vet hvordan de skal bruke algoritmer og regler for å løse et matematisk problem, men de har ikke forstått hvorfor algoritmene fungerer (Skemp, 1976, s. 2). Det motsatte av instrumentell forståelse i matematikk, er relasjonell forståelse. Relasjonell forståelse i matematikk kan beskrives som «kunnskap rik på relasjoner», det vil si at man ser sammenhenger mellom matematiske konsepter og begreper. Elever med en relasjonell forståelse vet derfor både hvordan og hvorfor en algoritme fungerer (Skemp, 1976, s. 2-3). Et eksempel på dette vil være å både kunne anvende divisjonsalgoritmen for å løse et divisjonsstykke, men også forstå hvorfor algoritmen fungerer.

Hiebert og Lefevre – Prosedural og konseptuell forståelse i matematikk

Hiebert og Lefevre (1986) har foreslått et liknende begrepspar på matematisk forståelse, men har valgt å disse for *prosedural* og *konseptuell* forståelse. Prosedural forståelse kan forklares gjennom to forhold: notasjon og algoritmer. *Notasjon* handler om å beherske det matematiske språket gjennom bruk av symboler. På et høyere matematisk nivå vil en slik type kunnskap være viktig for å kunne utføre bevis gjennom en streng symbolbruk (Hiebert & Lefevre, 1986,

s. 6). *Algoritmer* handler om at man klarer å bruke algoritmer og regler for å løse et matematisk problem. Dette er en type kunnskap som krever at man klarer å følge instruksjoner for å komme fram til et rett svar. Et sentralt trekk ved prosedural forståelse er ifølge Hiebert og Lefevre at man klarer å følge et sett med instruksjoner på en bestemt og tydelig måte. Det betyr at rekkefølgen man følger instruksjonene på, er avgjørende for at man skal lykkes. Konseptuell forståelse handler om å forstå hvordan matematiske konsepter henger sammen. Det betyr å se matematikk som «et koblet nettverk» hvor alt henger sammen. Konseptuell forståelse handler også om å bruke tidligere kunnskap i møte med nye problemer, på den måten utvikles ny matematisk kunnskap (Hiebert & Lefevre, 1986, s. 4).

Det Skemp kaller for *instrumentell forståelse* i matematikk har likhetstrekk med det Hiebert og Lefevre kaller for *prosedural forståelse* i matematikk. Det disse begrepene har til felles er de beskriver en type matematisk forståelse eller kunnskap som kan generere et rett svar, gitt at man følger algoritmene eller instruksene korrekt. Samtidig er de ulike på det punktet som har med notasjon å gjøre. For å ha prosedural forståelse i matematikk må man også beherske et mer abstrakt matematisk språk. Det Skemp kaller for *relasjonell forståelse* i matematikk er veldig likt det Hiebert og Lefevre kaller for *konseptuell forståelse*. Det disse begrepene har til felles er at de beskriver en type matematisk forståelse hvor man kan forstå hvordan matematiske konsepter, begreper og ideer henger sammen i et sammenkoblet nettverk. Elever som har en relasjonell og konseptuell forståelse i matematikk kan både bruke algoritmer og regler, men de kan også forklare hvorfor de fungerer.

Dermed kan begrepet algoritmisk tenkning også forstås i lys av Skemp og Hiebert og Lefevre sine begreper om matematisk forståelse. Grunnen til dette er fordi bruk av algoritmer i undervisningssammenheng har møtt kritikk fordi de fremmer en instrumentell forståelse i matematikk (Skemp,1976). Algoritmisk tenkning kan også forstås i lys av at elever må ha en prosedural forståelse i matematikk fordi algoritmer er abstraksjoner av den virkelige verden. Dette krever at elevene behersker et mer formelt matematisk språk (Hiebert og Lefevre, 1986). Algoritmisk tenkning kan også forstås i lys av relasjonell og konseptuell forståelse i matematikk, fordi ordet *tenkning* viser til at algoritmisk tenkning handler om noe mer enn å bare anvende standardalgoritmer i matematikk. Det må også innebære at elever forstår hvorfor en algoritme fungerer (Gjøvik & Torkildsen, 2019).

2.2.4 Hva er «tenkning» i programmering?

Et annet spørsmål er om *tenkning* er en egenskap som er forbeholdt oss mennesker, eller om det også er noe datamaskiner gjør. Ifølge Wing (2006, s. 35) er *tenkning* en menneskelig prosess. Wing skriver at datamaskiner i seg selv er kjedelige, men at det er vi mennesker som gjør de spennende. Med det mener hun at det er vi mennesker som skaper innholdet til en datamaskin og at datamaskinen bare er et verktøy for å skape, anvende og teste ut ideer. Det største arbeidet ligger i det å skape algoritmer på et programmeringsspråk som datamaskinen forstår, det er her *tenkningen* kommer inn. Denne oppgaven tar utgangspunkt i det samme som det Wing skriver, at tenkning er knyttet til en menneskelig egenskap.

Oppsummering

Algoritmisk tenkning viser seg å være et sammensatt begrep. I både matematikk og i programmering er algoritmer et sett med instrukser som løser et bestemt problem, hvor det i matematikkfaget løser matematiske problem, mens det i programmeringsfaget løser en rekke ulike problem (Bueie, 2019). Tenkning handler om å bearbeide mentale sanseinntrykk (Svartdal, 2018), og er ifølge Wing (2006) en ferdighet som er forbehold oss mennesker og ikke en datamaskin. Gjøvik og Torkildsen har foreslått å bruke ordet *tenkning* framfor *tankegang* når man snakker om algoritmisk tenkning. Dette er fordi man i matematikkfaget er opptatt av generelle måter å arbeide og å tenke på. Denne oppgaven tar utgangspunkt i å forstå tenkning i lys av matematisk forståelse jf. Skemp (1976) og Hiebert og Lefevre (1986) sine begreper.

Noen foreløpige slutninger er at algoritmisk tenkning fremstår som et gammelt begrep i en ny kontekst. Algoritmer er ikke noe nytt i matematikkfaget, men de ble enda mer betydningsfulle når de kunne anvendes på en datamaskin. Dermed kan det hende at det er ordet tenkning som bidrar til at begrepet oppleves som noe nytt. Ikke bare må man forstå hvordan en algoritme fungerer, men man må også forstå hvorfor den fungerer. Dette vil være viktig både i matematikkfaget og i programmeringsfaget.

2.3 Del 2: Hvordan brukes begrepet i internasjonal og norsk forskningslitteratur?

Ifølge Gjøvik og Torkildsen (2019, s. 32) er det uoverensstemmelse om hvordan begrepet algoritmisk tenkning defineres og brukes i internasjonal og norsk forskningslitteratur. Dermed vil jeg i teorikapittelet del to gå nærmere inn på dette.

2.3.1 I internasjonal forskningslitteratur

Ifølge Bueie (2019, s. 28) kom begrepet algoritmisk tenkning for alvor fram i lyset gjennom den populærvitenskapelige artikkelen «*Computational thinking*», skrevet av Jeanette Wing i 2006 (Wing, 2006). Jeanette Wing er professor i informatikk ved Columbia University og har fått mye oppmerksomhet for å løfte fram dette begrepet i skolesammenheng. Artikkelen har ifølge Bueie blitt et viktig referansepunkt i debatten rundt innføringen av programmering i matematikkfaget. I Wing sin artikkel brukes begrepet *computational thinking* hvor *algorithmic thinking* blir sett på som en viktig del av computational thinking. Videre vil computational thinking forkortes til CT.

Ifølge Wing kan CT defineres slik:

> Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science (Wing, 2006, s. 33).

Som man kan se er dette en ganske overordnet og vid definisjon av CT-begrepet. En av grunnene til dette er at Wing ser på CT som en universell ferdighet som ikke er avgrenset til bare ett fag, som for eksempel matematikkfaget. Hun skriver følgende:

> It (altså CT) represents a univsersally applicable attitude and skill set everyone, not just computer scientist, would be eager to learn and use (Wing, 2006, s. 33).

Wing mener at CT burde implementeres på alle skoler, på alle trinn og i alle fag. Videre skriver hun at CT innebærer ferdigheter i problemløsning, abstraksjon, algoritmisk tenkning, modellering og dekomponering. Ifølge Bueie (2019, s. 26) er det er lett å kjenne igjen matematikken fra disse stikkordene. Kanskje er dette noen av grunnene til at man drar paralleller fra CT-begrepet til matematikkfaget? Wing bruker en mindre del av sin artikkel på

å si hvorfor CT kan være viktig for matematikkfaget, men det hun skriver er at CT og matematikk utfyller hverandre på en god måte fordi programmeringsfaget har sitt utspring fra matematikken. Bocconi et al. (2016) skriver at Wing sin artikkel om CT har skapt store diskusjoner om hva CT egentlig er fordi definisjonene hennes har vært for vide. Wing kom derfor i 2011 med en ny definisjon av hva CT er. Hun skriver følgende:

> Computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent (Wing referert i Bocconi et al., 2016, s. 15).

Bocconi et al. skriver at det som er interessant med Wing sin nye definisjon er at hun ikke skriver noe om at «*information-processing agent*», må være en datamaskin. Bocconi et al. tolker dermed Wing sin definisjon til å bety følgende:

1) CT er en tankeprosess, men den kan utvikles uavhengig av teknologi.

2) CT er en egen type problemløsningsmetode som innebærer ulike typer ferdigheter. Ett av disse er å kunne utvikle algoritmer som kan utføres av enten en datamaskin, et menneske eller av begge.

Bocconi sin tolkning av Wing sin definisjon er interessant for matematikkfaget. Ved å si at CT kan være uavhengig av teknologi, men heller handle om en egen måte å tenke på, kan man som matematikklærere stå friere i valget om man vil bruke et programmeringsverktøy eller ikke for å utvikle elevers algoritmiske tenkning. På den andre siden er det som Wing skriver, naturlig å dra paralleller fra matematikkfaget til programmeringsfaget fordi programmering har sitt utspring fra matematikken. Dermed er det kanskje ikke så rart at matematikkfaget har fått en sentral rolle i å lære opp elever i programmering (Bueie, 2019, s. 23).

Grover og Pea (2013, s. 1) skriver at det er vanlig å referere tilbake til Wing sin artikkel i 2006 som startskuddet for CT-begrepet, men at begrepet egentlig stammer ifra matematikere Seymour Papert i 1996. Seymour Papert skrev i 1980 en bok som han kalte *Mindstorms: Children, computers, and powerful ideas*, som av mange ble sett på som et paradigmeskifte innen pedagogikken (Drijvers et al., 2009, s. 91-92). Seymour Paper var inspirert av et konstruktivistisk læringssyn hvor han mente at elever ville oppdage matematikken på en ny måte gjennom å programmere. I stedet for at elever skulle være «brukere» av matematikken,

skulle de nå være med på å skape matematikken. Grover og Pea (2013, s. 39) skriver at Papert brukte CT-begrepet synonymt med prosedural tenkning. Papert mente at elever ville lære seg å kommunisere matematikk på en logisk måte gjennom å fortelle en datamaskin hva den måtte gjøre.

Her vil det være naturlig å trekke inn Hiebert og Lefevre (1986) sitt begrep om prosedural forståelse i matematikk. Hiebert og Lefevre beskriver prosedural forståelse som en viktig ferdighet for å klare å følge algoritmer nøyaktig, men også en ferdighet i å beherske et mer abstrakt matematisk språk. I en programmeringskontekst kan prosedural forståelse handle om å klare å «oversette» det matematiske språket til et programmeringsspråk.

I internasjonal forskningslitteratur er det altså CT-begrepet som brukes, her er *algorithmic thinking* en del av CT. Wing skriver at CT kan utvikles både med og uten teknologi, og at det er en egen type problemløsningsmetode som innebærer ulike typer ferdigheter.

2.3.2 I norsk forskningslitteratur

I Norge har vi valgt å oversette CT-begrepet direkte til algoritmisk tenkning (Gjøvik & Torkildsen, 2019, s. 32). Et utfordring med dette, er at vi i Norge mangler et tilsvarende norsk begrep for det som på engelsk kalles for *algorithmic thinking*. I internasjonal forskningslitteratur er algorithmic thinking en viktig del av CT-begrepet, men i Norge mangler vi et slikt begrep. En mulighet hadde vært å byttet ut CT-begrepet med beregningstankegang, hvor algoritmisk tenkning var en viktig del av dette.

Gjøvik og Torkildsen (2019, s. 33) forslår at man kan bruke begrepet *algoritmebehandling* som en erstatning for algorithmic thinking. I deres artikkel handler algoritmebehandling om det å følge og forklare trinnvise instruksjoner. Dette er det samme begrepet som brukes videre i denne oppgaven.

Følgende får man at:

- Algorithmic thinking er en del av computational thinking (internasjonal forskningslitteratur)
- Algoritmebehandling er en del av algoritmisk tenkning (norsk forskningslitteratur).

I norske læreplaner

På utdanningsdirektoratet sine nettsider under *kvalitet og kompetanse – profesjonsfaglig digital kompetanse*, står det følgende om algoritmisk tenkning:

> Algoritmisk tenkning er en problemløsningsmetode. Algoritmisk tenkning innebærer å tilnærme seg problemer på en systematisk måte, både når vi formulerer hva det er vi ønsker å løse og når vi foreslår mulige løsninger. Litt forenklet kan vi si at det er «å tenke som en informatiker» når vi skal løse problemer eller oppgaver (Utdanningsdirektoratet, 2020a, s. 1).

Her skriver utdanningsdirektoratet at algoritmisk tenkning er en problemløsningsmetode og at det handler om å tilnærme seg problemer på en systematisk måte. En kommentar til dette er om ikke denne definisjonen blir for enkel? Kjennetegnes ikke alle problemløsningsmetoder av at man prøver å tilnærme seg et problem på en systematisk måte? Hvordan skiller man *denne* problemløsningsmetoden fra andre typer problemløsningsmetoder? Videre skriver utdanningsdirektoratet at algoritmisk tenkning handler om «å tenke som en informatiker». Det er litt uklart hva som menes med akkurat dette. Hvordan skal lærere vite hva som er særegent med å tenke som en informatiker? En mulig forklaring på dette kommer litt lengre ned på utdanningsdirektoratet sine nettsider:

> Å tenke algoritmisk er å vurdere hvilke steg som skal til for å løse et problem, og å kunne bruke sin teknologiske kompetanse for å få en datamaskin til å løse (deler av) problemet (Utdanningsdirektoratet, 2020a, s. 1).

Dette kan minne om hvordan Bocconi et al. (2016) har tolket Wing sin definisjon av CT-begrepet, nemlig at algoritmisk tenking er en tankeprosess som kan utvikles både med og uten bruk av teknologi. Utdanningsdirektoratet skriver at det finnes flere definisjoner av algoritmisk tenkning, men at de har valgt å dele algoritmisk tenkning inn i seks nøkkelbegrep og fem arbeidsmåter. Nøkkelbegrepene er: *logikk, algoritmer, dekomposisjon, mønstre, abstraksjon og evaluering.* Arbeidsmåtene er: *fikle, skape, feilsøke, holde ut og samarbeide.* I

undervisningssammenheng er det fint å kunne ta i bruk noen konkrete begreper, men jeg har ikke klart å finne noe mer utfyllende informasjon som forklarer disse begrepene på et utdypende måte.

I læreplanen for matematikk

Algoritmisk tenkning blir også beskrevet i læreplanen for matematikk under kjerneelementet *utforsking og problemløsning*:

> Algoritmisk tenkning er viktig i prosessen med å utvikle strategier og framgangsmåter for å løse et problem og innebærer å bryte ned et problem i delproblem som kan løses systematisk. Videre innebærer det å vurdere om delproblemene best kan løses med eller uten digitale verktøy (Utdanningsdirektoratet, 2020c, s. 2).

Her står det at algoritmisk tenkning handler om å løse et problem på en systematisk måte gjennom å bryte det ned til mindre delproblem. I motsetningen til den andre definisjonen til utdanningsdirektoratet, står det ingenting her om at algoritmisk tenkning er en problemløsningsmetode. Det legges heller mer vekt på at algoritmisk tenkning er en del av en større prosess som handler om å utvikle strategier og framgangsmåter for å løse et problem. Det som disse definisjonene allikevel har til felles, er at algoritmisk tenkning kan utvikles både med og uten bruk av digitale verktøy.

I læreplanmålene fra 5.-10. trinn er det kun i 5.trinn og 8. trinn hvor ordet algoritme brukes eksplisitt. For 5. trinn står det: «Lage og programmere algoritmer med bruk av variabler, vilkår og løkker». Og for 8. trinn står det: «Utforske hvordan algoritmer kan skapes, testes og forbedres ved hjelp av programmering» (Utdanningsdirektoratet, 2020c, s. 35-37). Begge disse kompetansemålene kan oppleves som litt tekniske da de ikke har noen direkte kobling til matematikkfaget. Kaufmann og Forsström (2019, s. 7-8) skriver at det ikke er den tekniske siden ved programmering som er den mest interessante, men heller hvordan man kan bruke programmering for å utvikle elevers tenkning og problemløsningsferdigheter i matematikk. Dette støttes av Gjøvik og Torkildsen som skriver at programmering må være noe mer enn å bare implementere allerede eksisterende algoritmer. Målet bør være læring *av* matematikk men *med* programmering (Gjøvik & Torkildsen, 2019, s. 36).

Oppsummering

I internasjonal og norsk forskningslitteratur ser man at algoritmisk tenkning defineres og brukes forskjellig (Bocconi et al., 2016; Gjøvik & Torkildsen, 2019; Utdanningsdirektoratet, 2020a, 2020b; Wing, 2006, 2008). Det som ser ut til å være sammenfallende mellom hvordan begrepet brukes i internasjonale og norske termer er følgende:

1) Computational thinking eller algoritmisk tenkning kan utvikles både med og uten teknologi (Bocconi et al., 2016; Utdanningsdirektoratet, 2020c; Wing, 2006).

2) Computational thinking eller algoritmisk tenkning er både en tankeprosess og en problemløsningsmetode som innebærer å tilnærme seg et problem på en systematisk måte gjennom å bryte problemet ned i mindre delproblemer (Utdanningsdirektoratet, 2020; Wing, 2006).

Jeg har ikke lyktes i å finne noen norske artikler som skriver om sammenhengen mellom algoritmisk tenkning og prosedural kunnskap selv om disse to begrepene ble brukt synonymt med hverandre av Seymour Papert på 1960-tallet (Paper referert i Grover & Pea). En viktig del av prosedural forståelse i matematikk er å kunne beherske det matematiske språket gjennom bruk av symboler (Hiebert & Lefevre, 1986). Dermed vil det være naturlig at også programmering vil kreve prosedural forståelse fordi man må oversette et matematisk språk til et programmeringsspråk. Dermed kan det tenkes at tredje punkt er:

3) Computational thinking eller algoritmisk tenkning krever prosedural forståelse i både matematikk og i programmering.

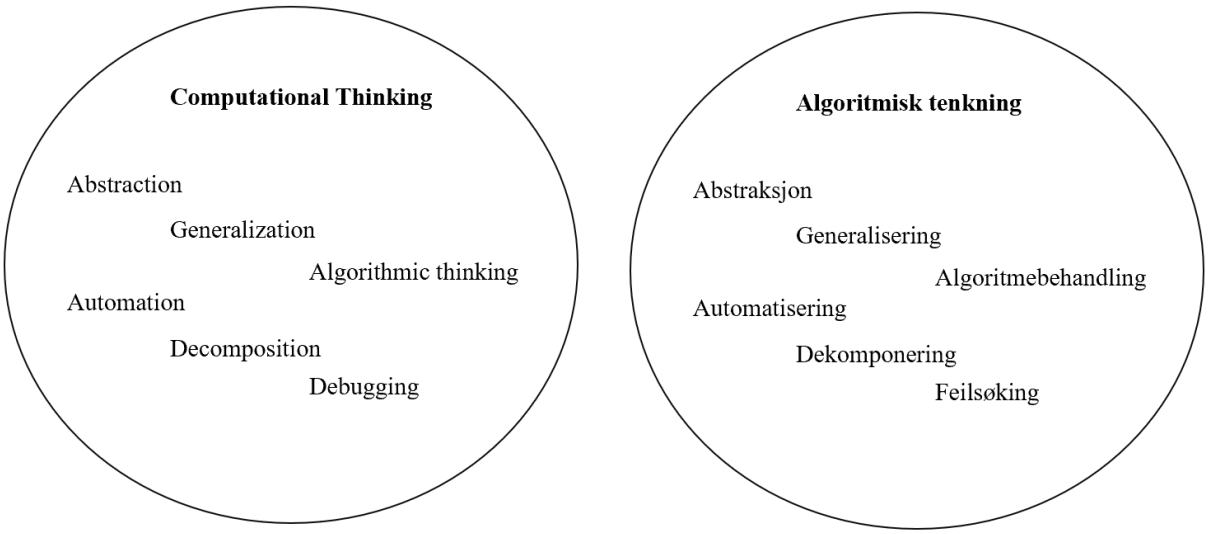
2.4 Del 3: Hva kjennetegner algoritmisk tenkning?

Problemstillingen for denne masteroppgaven er: *Hva kjennetegner elevers algoritmiske tenkning når de jobber med geometri i blokkprogrammering?* Etter å ha undersøkt hva algoritmisk tenkning er og hvordan begrepet brukes i internasjonal og norsk forskningslitteratur, handler del 3 av teorikapittelet om hva som kjennetegner algoritmisk tenkning.

2.4.1 Bocconi et al. (2016): Et rammeverk for *computational thinking*

I delkapittel 2.2.3 står det at algoritmisk tenkning kan forstås gjennom Skemp (1976) og Hiebert og Lefevre (1986) sine begreper om matematisk forståelse. Dette er fordi ordet *tenkning* er for abstrakt til at det kan måles direkte hos elever. Allikevel vil begrepene til Skemp og Hiebert og Lefevre ha sine begrensninger for denne oppgaven da de ikke er skrevet i sammenheng med programmeringsfaget. En annen mulighet er å bruke nøkkelbegrepene og arbeidsmåtene til utdanningsdirektoratet. Utfordringer her er manglende informasjon om hvordan man skal tolke disse begrepene. Dermed tar denne oppgaven utgangspunkt i et internasjonalt rammeverk som er skrevet av Stefania Bocconi, Augusto Chioccariello, Giuliana Dettori og Anusca Ferrari (2016): *Developing Computational thinking in Compulsory Education. Implications for policy and practice.* Dette rammeverket er ikke utviklet i en matematikkontekst men på et politisk nivå. Det vil si at hensikten med rammeverket var å få en felles konsensus rundt CT-begrepet når programmering skal inn i skolen.

Bocconi et al. (2016) har gjennom en større litteraturanalyse kommet fram til seks kjennetegn på CT. Disse er: *abstraction, generalization, algorithmic thinking, decomposition, automation,* og *debugging.* Gjøvik og Torkildsen (2019) skriver at når vi i Norge har valgt å oversette CT til algoritmisk tenkning må det bety at algoritmisk tenkning består av de samme kjennetegnene. Gjøvik og Torkildsen har oversatt Bocconi et al. (2016) sine begrepet til norsk:



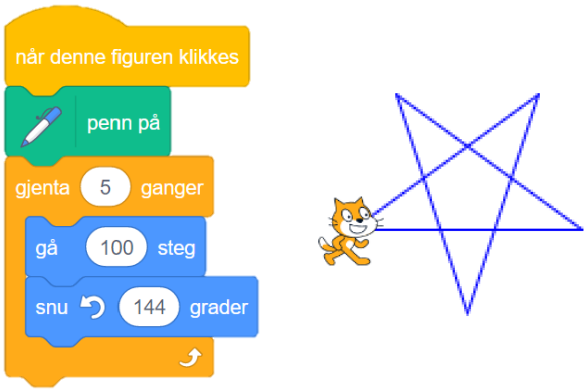
Figur 3 Bocconi et al. (2016) sine seks kjennetegn på computational thinking (figur t.v). Gjøvik og Torkildsen (2019) sine norske oversettelser (figur t.h).

Figuren til venstre viser begrepene i Bocconi et al. (2016) sitt rammeverk. Figuren til høyre viser Gjøvik og Torkildsen (2019) sine norske oversettelser. Som man kan se er *algorithmic thinking* byttet ut med *algoritmebehandling*.

I denne oppgaven har jeg valgt å bare ta utgangspunkt i kjennetegnene abstraksjon, generalisering og algoritmebehandling. Grunnen til dette er at disse kjennetegnene var mest relevant for hvordan jeg utformet oppgaven til elevene. For videre lesning har blant annet Kaufmann og Stenseth (2020) utforsket *feilsøking* gjennom å se på ulike typer feilrettingsforsøk hos elever som programmerer. For å gjøre Bocconi et al. (2016) sitt rammeverk mer relevant for min studie, vil jeg knytte disse tre kjennetegnene til matematikkfaget. Videre kommer jeg til å bruke Gjøvik og Torkildsen sine norske oversettelser.

Abstraksjon

Abstraksjon er ifølge Bocconi et al. (2016, s. 18) det viktigste kjennetegnet på algoritmisk tenkning fordi programmering krever at man klarer å lage abstraksjoner av den virkelige verden på et språk som datamaskinen forstår. Bocconi et. al skriver at abstraksjon handler om å gjøre et konsept mer forståelig gjennom å redusere unødvendige detaljer. Også Wing (2008, s. 3717) trekker fram abstraksjon som selve essensen bak algoritmisk tenkning. Et eksempel på abstraksjon er figuren som ble presentert tidligere i denne oppgaven:



Figur 4 : Algoritmen til venstre er en abstraksjon av en femkantet stjerne. Laget i Scratch.

.

Algoritmen til venstre er en abstraksjon av en stjerne fordi disse kodelinjene er uttrykt på et språk som datamaskinen skal kunne lese. Stjernen til høyre er en grafisk framstilling av det som står i algoritmen. Dermed er figuren til høyre en konkret og fysisk representasjon av en stjerne, mens algoritmen til venstre er en abstrakt representasjon av en stjerne.

Ifølge Davis et al. (2011, s. 121) er abstraksjon selve «blodet» i matematikkfaget. Grunnen til dette er at man i matematikk har med tenkte objekter å gjøre. Abstraksjon i matematikk handler om å uttrykke matematiske ideer på en enklere måte ved å se bort ifra irrelevante opplysningen og fokusere på det som er viktigst. Dette har likhetstrekk med det Bocconi et al. skriver om at man i programmering ønsker å gjøre et konsept mer forståelig gjennom å redusere unødvendige detaljer. Abstraksjon er også et av kjerneelementene i fagfornyelsen 2020, hvor det står at abstraksjon i matematikk innebærer at elever går fra å bruke et mer konkret matematisk språk til å gradvis utvikle et mer formelt symbolspråk og formelle resonnement (Utdanningsdirektoratet, 2020c, s. 31).

Wing (2008, s. 3717) skriver at abstraksjoner i programmering er mer generelle enn abstraksjoner i matematikk. En av grunnene til dette er at matematiske konsepter kan uttrykkes på en enkel måte gjennom for eksempel algebraisk notasjon. Et eksempel på dette kan være å uttrykke et partall på formen $2c$. Symbolet c representerer et hvilket som helst tall. Man forsøker med andre å trekke ut en felles egenskap fra en samling av objekter.

Abstraksjoner i programmering bli dermed mer komplekse enn abstraksjoner i matematikk fordi en kode kan brukes i mange ulike sammenhenger.

Generalisering

Generalisering handler ifølge Bocconi et al. (2016, s. 18) om å oppdage og identifisere mønstre og likheter i et datasett. Mønstergjenkjenning er dermed en viktig forløper for å generalisere. En viktig del av å lage generaliserte algoritmer handler om å spørre seg: «er dette problemet likt et problem jeg har løst før»? Ved å utvikle algoritmer som kan generaliseres kan man få datamaskinen til å løse et sett med liknende problem.

Generalisering i matematikk henger sammen med abstraksjon, fordi man generaliserer over abstrakte objekter (Davis et al., 2011, s. 150). Uttrykket for partall, *2c* er et eksempel på både en abstraksjon og en generalisering. Symbolet *c* er abstraksjonen, men hele uttrykket kan generaliseres for alle partall. Generalisering er også et av kjerneelementene i fagfornyelsen 2020, hvor det står at generalisering innebærer at elever oppdager sammenhenger , strukturer og mønstre i matematikk (Utdanningsdirektoratet, 2020c, s. 31).

Algoritmebehandling

Algoritmebehandling handler ifølge Bocconi et al. (2016, s. 31) om å kunne skape, implementere og vurdere algoritmer. En algoritme må være skrevet på en stegvis og korrekt måte for at datamaskinen skal kunne omgjøre algoritmen til en ønsket handling.

Algoritmebehandling kan i likhet med algoritmisk tenkning, forstås gjennom Skemp (1976) og Hiebert og Lefevre (1986) sine begreper om instrumentell, relasjonell, prosedural og konseptuell forståelse i matematikk. Hvis elever bare følger eller bruker algoritmer uten å forstå hva de gjør eller hvorfor de fungerer, kan det knyttes til instrumentell forståelse (Skemp, 1976). Hvis elever klarer å oversette et matematisk språk til en programmeringsspråk kan det knyttet til prosedural forståelse (Hiebert og Lefevre, 1986; Grover & Pea, 2013). Hvis elever forstår hvorfor en algoritme fungerer og hvorfor den løser et matematisk problem kan det knyttes til relasjonell og konseptuell kunnskap (Hiebert & Lefevre, 1986; Skemp, 1976).

Algoritmebehandling blir ikke eksplisitt nevnt i fagfornyelsen fordi dette er et relativt nytt ord som Gjøvik og Torkildsen (2019) har foreslått som en erstatning for *algorithmic thinking.* Men under kjerneelementet *resonnering og argumentasjon*, står det at elever skal kunne

følge, vurdere og forstå matematiske tankerekker, noe som innebærer at de også skal kunne forstå matematiske regler (Utdanningsdirektoratet, 2020c, s. 31).

Abstraksjon, generalisering og *algoritmebehandling* er dermed de tre kjennetegnene som brukes for å analysere hvordan elevene løser oppgaven de får utdelt.

Kapittel 3

3 Metode

Metodekapittelet er en systematisk redegjørelse som viser hvordan man som forsker har gått fram for å finne svar på problemstillingen. Problemstillingen for denne oppgaven er: *Hva kjennetegner elevers algoritmiske tenkning når de arbeider med geometri i blokkbasert programmering?* I dette kapittelet vil jeg presentere 1) studiens kunnskapssyn, 2) valg av metode og forskningsdesign, 3) metode for datainnsamling, 4) forberedelse og gjennomføring av studien, 5) analysemetode og 6) kvalitet i studien.

3.1 Studiens kunnskapssyn

Et kunnskapssyn sier noe om hvordan man som forsker kan frembringe sann eller troverdig kunnskap om fenomenet som studeres (Postholm, Jacobsen & Søbstad, 2018, s. 45). Denne studien undersøker hva som kjennetegner elevers algoritmiske tenkning med utgangspunkt i Bocconi et. al (2016) sine tre kjennetegn på algoritmisk tenkning: abstraksjon, generalisering og algoritmebehandling. Creswell (2014, s. 8) skriver at dersom forskeren ønsker å frembringe kunnskap om et fenomen gjennom å se på deltakernes *tolkning* av fenomenet, har forskeren et konstruktivistisk kunnskapssyn. Denne studien passer til Creswell sin beskrivelse, fordi det er elevenes tolkninger av oppgaven som utgjør studiens empiri. At studien har et konstruktivistisk vitenskapssyn gjenspeiles også i problemstillingen for denne oppgaven. Creswell skriver at man med et konstruktivistisk læringssyn ofte velger vide problemstillingen slik at deltakerne kan konstruere sin egen mening ut av det som studeres.

3.2 Valg av metode

Valg av metode henger sammen med studiens kunnskapssyn og studiens problemstilling. Med en kvantitativ tilnærming til problemstillingen kunne man ha sett hvor ofte Bocconi sine kjennetegn på algoritmisk tenkning opptrådde i datamaterialet og deretter forsøkt å tallfeste disse kjennetegnene for å si noe generelt om hva som kjennetegner elevers algoritmiske tenkning. Én av utfordringene med å velge en kvantitativ metode er kravet om operasjonalisering, det vil si at man må gjøre et abstrakt begrep operativt for å kunne måle det (Postholm et al., 2018, s. 168). Algoritmisk tenkning er et abstrakt begrep, og forskning viser at det fortsatt er uenigheter i hvordan man skal tolke og forstå dette begrepet (Gjøvik &

Torkildsen, 2019). I tillegg er de tre kjennetegnene *abstraksjon, generalisering* og *algoritmebehandling* ganske vide, noe som kan gi rom for fortolkninger hos meg som forsker når datamaterialet skal analyseres. På bakgrunn av dette valgte jeg en kvalitativ metode for å besvare oppgavens problemstilling. Kvalitative metoder kjennetegnes av rike beskrivelser av et mindre utvalg, hvor man søker å gå i dybden på et tema eller et fenomen som man som forsker ikke har tilstrekkelig med kunnskap eller kjennskap til (Ryan, 2006, s. 21).

3.2.1 Instrumentell casestudie

Kvalitative metoder er et samlebegrep for ulike forskningsdesign og tilnærminger som kjennetegnes av fleksibilitet og åpenhet (Postholm et al., 2018, s. 137; Thagaard, 2013, s. 55). Et av disse er casestudie, som er et type forskningsdesign hvor man ønsker å få en grundigere forståelse for noen få enheter eller caser (Creswell, 2013, s. 247).

Min studie går inn under det Creswell (2013, s. 98) kaller for en instrumentelt casestudie. I en instrumentell casestudie utforskes fenomenet i studien gjennom å studere en enkelt enhet (Thagaard, 2013, s. 56). Creswell skriver at med en instrumentell casestudie legges det mindre vekt på personene som deltar i studien og mer vekt på fenomenet som studeres. Personene som deltar i studien utgjør dermed det empiriske grunnlaget for å utvikle forståelse for fenomenet (Creswell, 2013, s. 99-104). En instrumentell casestudie passer godt til oppgavens problemstilling og hensikten med dette studie. Hensikten med denne studien er å utvikle forståelse for hva som kjennetegner algoritmisk tenkning og dette undersøkes gjennom å studere 10 elever.

Dermed er denne studien en kvalitativ, instrumentell casestudie som er forankret i et konstruktivistisk vitenskapssyn.

3.2.2 Videoopptak som metode for datainnsamling

Hensikten med denne studien er å analysere elevers algoritmiske tenkning. Dermed måtte jeg velge en datainnsamlingsmetode som kunne gi meg rike beskrivelser. Silverman (2011) skriver i boken *Interpreting qualitative data*, om fire ulike kvalitative framgangsmåter for å samle inn data. Disse er: 1) observasjon 2) intervju 3) analyse av foreliggende tekster og visuelle uttrykksformen og 4) analyse av lyd – og videopptak. På bakgrunn av at jeg ønsket å analysere hva elevene både gjorde og sa, valgte jeg videoopptak som metode for datainnsamling.

Videoopptak som metode for datainnsamling går inn under det Dicks, Mason, Coffey og Atkinson (2006, s. 71) kaller for *visuelle data som forskeren selv produserer i feltarbeidet*. Det vil si data som forskeren selv har initiert å bruke gjennom for eksempel bruk av foto eller lyd- og videoopptak. Thagaard (2013, s. 143) skriver at fordelen med å bruke videoopptak er at man som forsker har muligheten til å studere både verbal og non-verbal kommunikasjon. Det vil si at man har mulighet til å studere både det man hører og det man ser. En annen fordel er at man i etterkant av datainnsamlingen har mulighet til å gjennomgå videopptakene flere ganger.

Jeg fikk låne fem GoPro-kameraer av universitetet i Tromsø. Disse kameraene hadde i tillegg egne brystbelter og hodestropper slik at elevene som deltok i studien kunne feste GoPro-kameraene på seg.

3.2.3 Observasjon som metode for datainnsamling

Ifølge Adler og Adler (1994) er observasjon den mest grunnleggende metoden å samle inn data på i kvalitativ forskning. Det vanligste er å gjennomføre observasjon i naturlige settinger slik at deltakerne får utspille seg i en naturlig kontekst.

I denne studien inntok jeg en «fullstendig observatørrolle» (Gold, 1957). En fullstendig observatørrolle kjennetegnes av at man som forsker har stor avstand og liten deltakelse til de som deltar i studien (Thagaard, 2013, s. 115). Som fullstendig observatør deltar man ikke i aktiviteten som observeres og man skal i den grad det er mulig, ikke samhandle med de som observeres. Man har allikevel lov til å svare på spørsmål som er av praktisk betydning (Thagaard, 2013, s. 115). Savin-Baden og Howell-Major (2013) beskriver dette som *den passive deltakerrollen*. Man befinner seg i settingen som observeres, men man forsøker å påvirke den minst mulig.

Jeg valgte å kombinere videoopptak med en fullstendig observatørrolle. Det vil si at jeg befant meg i samme rom som elevene men jeg forsøkte å minimere kontakten så mye som mulig. Jeg presenterte oppgaven for elevene men inntok deretter en passiv deltakerrolle hvor jeg satt med avstand til elevene. Ved å bruke videoopptak kunne jeg også gjennomgå opptakene i etterkant av studien.

3.3 Utvelgelse av informanter

Christoffersen og Johannessen (2012, s. 49) beskriver ulike utvalgsstrategier når man skal velge informanter til sitt forskningsprosjekt. De skriver at utgangspunktet for utvelgelse av informanter ved kvalitative metoder, er å velge ut informanter basert på hensiktsmessighet og ikke representativitet. Grunnen til dette er at kvalitative metoder kjennetegnes av at man forsøker å få rike beskrivelser av et begrenset antall personer. En av utvalgsstrategiene som Christoffersen og Johannessen beskriver er *kriteriebasert utvelgelse.* Det betyr at informantene velges ut på bakgrunn av noen kriterier som man som forsker har oppgitt.

Jeg hadde oppgitt to kriterier for deltakelse på denne studien. Det første kriteriet var at informantene måtte gå i 6. klasse. Grunnen til dette var fordi oppgaven som elevene fikk utdelt var inspirert av et kompetansemål for 6. trinn. Det andre kriteriet var at elevene måtte være komfortable med å bli tatt videoopptak av. Jeg kom i kontakt med to skoler som ønsket å delta på studien. Grunnen til at jeg kontaktet to skoler var fordi det ble problemer med filmingen første gang jeg gjennomførte studien. Jeg kommer til å si mer om dette i neste delkapittel.

3.4 Forberedelse og gjennomføring av studien

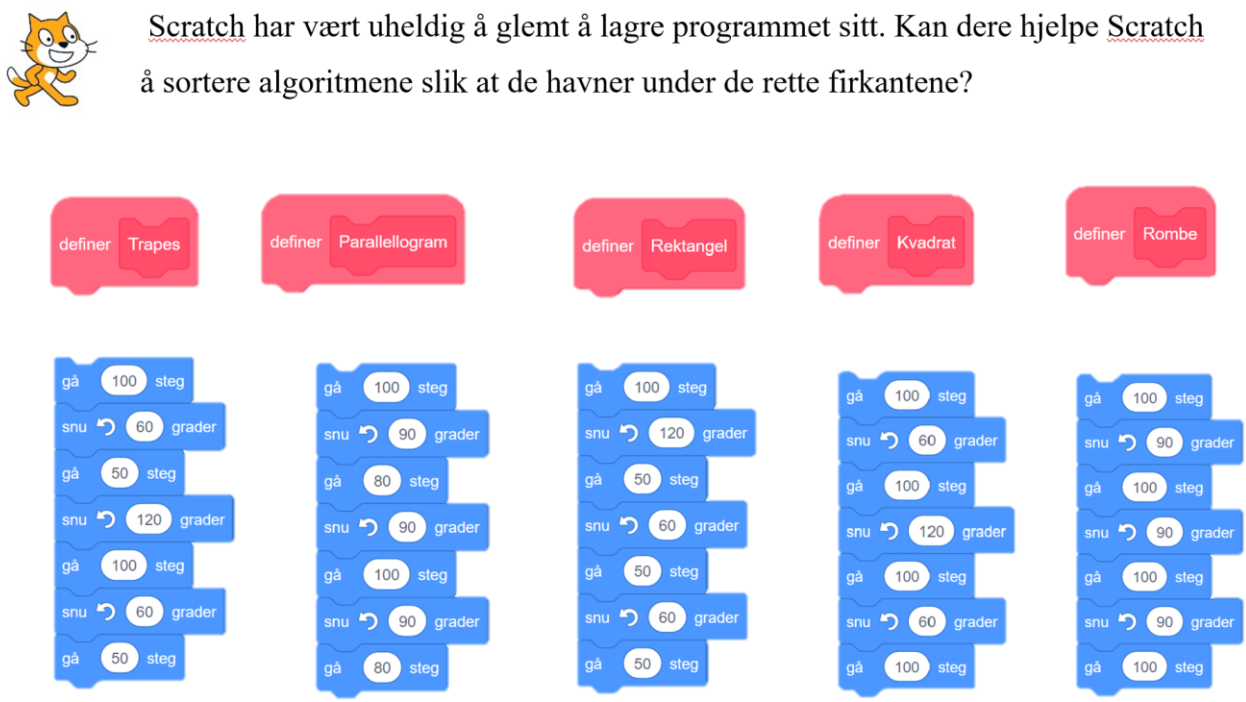
3.4.1 Utforming av oppgaven

Jeg ønsket å lage en oppgave som lå i skjæringspunktet mellom matematikkfaget og programmeringsfaget, men som ikke krevde forkunnskaper i programmering. Allikevel måtte oppgaven «invitere» til algoritmisk tenkning, samt inneholde noen viktige programmeringskonsepter.

Oppgaven er inspirert av en undervisningsform som kalles for Use-Modify-Create (UMC), som har som formål å utvikle elevers algoritmiske tenkning (Lytle et al., 2019, s. 9). Tanken bak UMC er at elever kan ta i bruk ferdiglagde algoritmer og programmer i stedet for å programmere dem selv. Dette kan være gunstig i starten dersom elevene ikke har kjennskap til programmering fra før. På den måten kan man unngå at det tekniske overskygger det faglige innholdet i undervisningsøkten. Elevene som skulle delta i denne studien hadde lite erfaring med programmering fra før, derfor valgte jeg å programmere fem algoritmer i Scratch som elevene skulle tolke og analysere. Oppgaven er også inspirert av et kompetansemål i geometri for 6. trinn:

Beskrive egenskaper og minimumsdefinisjoner ved to- og tredimensjonale figurer og forklare hvilke egenskaper figurene har til felles, og hvilke egenskaper som skiller figurene fra hverandre» (Utdanningsdirektoratet, 2019, s. 9).

Oppgaven tar bare utgangspunkt i firkanter. Her er oppgaven:



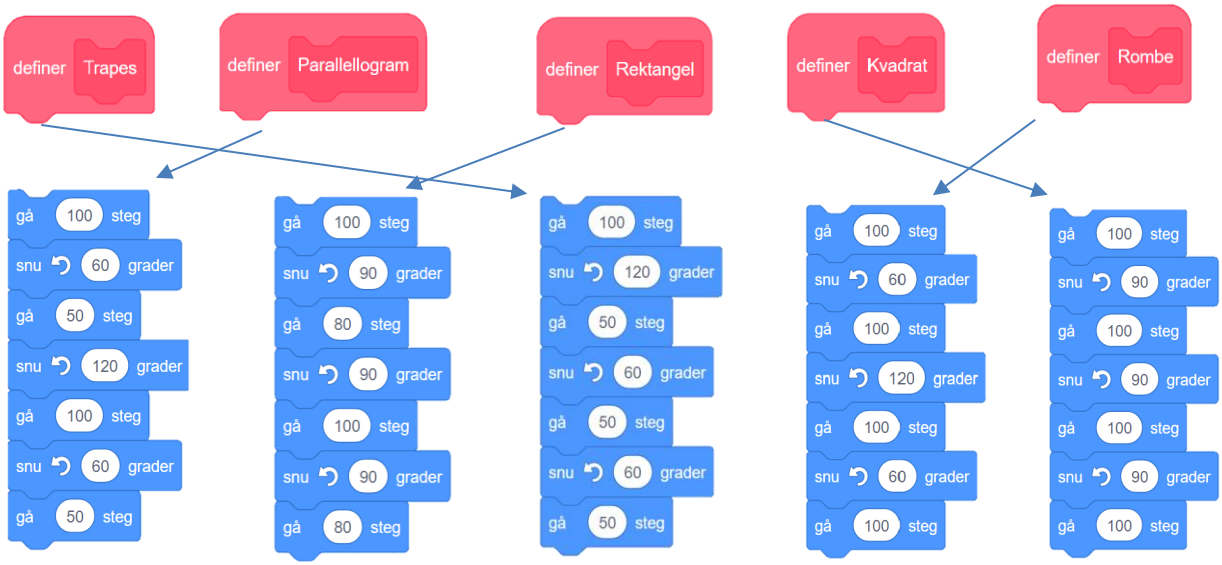
Figur 5: Oppgaven som elevene fikk. Algoritmene er laget i scratch.mit.edu

3.4.2 Begrunnelse for valg av oppgave

Bueie (2019, s. 28) skriver at en algoritme i programmering er en ordnet liste med instrukser som viser hvordan du må gå fram for å løse et bestemt problem. På bakgrunn av dette kan man si at oppgaven oppfyller kravet om å være en algoritme i programmering.

Tanken bak oppgaven var å se hvordan kjennetegnene abstraksjon, generalisering og algoritmebehandling kom til uttrykk gjennom hvordan elevene løste oppgaven. Alle algoritmene er abstraksjoner av to matematiske konsepter, hvor *gå-blokken* representerer et linjestykke, mens *snu-blokken* representerer en rotasjon eller en ytre vinkel. Alle algoritmene har det til felles at de er firkanter, noe man kan se gjennom at *gå 100-steg* blokken gjentar seg i alle algoritmene (generalisering). Alle algoritmene er ordnede lister med instruksjoner som

viser hvordan elevene må gå fram for å løse problemet (algoritmebehandling). Til slutt vil jeg presisere at ingen av algoritmene inneholdt noen feil. Her er fasiten på oppgaven:



Figur 6: Fasit på oppgaven. Pilene viser hvilken algoritme som hører til den rette firkanten.

Teststudie

Etter godkjennelse fra NSD fikk jeg låne 5 GoPro-kameraer av universitetet i Tromsø. Studien ble deretter gjennomført i en 6.klasse i Tromsø med 10 elever. På dette tidspunktet hadde jeg utformet oppgaven digitalt. Det vil si at elevene skulle arbeide med å sortere algoritmene på nettsiden til Scratch i stedet for på papir. I tillegg hadde jeg valgt å filme alle de 10 elevene samtidig mens de løste oppgaven. Det var flere ting som ikke gikk som planlagt og som gjorde at studien måtte gjennomføres på nytt.

Det ene var at det ble for kaotisk å filme 10 elever samtidig mens de arbeidet i Scratch. Noen av elevene fikk oppstartsproblemer med nettsiden, mens andre fikk problemet med nettilgang. Dermed ble mye av tiden brukt på å at elevene skulle komme seg i gang. Samtalene mellom elevene ble derfor veldig tekniske og lite interessant for oppgavens problemstilling.

På bakgrunn av dette måtte jeg finne en ny klasse som kunne delta på studien. Jeg kontaktet NSD og fikk lov til å bruke det samme informasjonsskrivet på en ny klasse. Dermed vil man se at informasjonsskrivet (vedlegg 2) er annerledes enn hvordan studien faktisk ble

gjennomført. I stedet for å undersøke hvordan elevene argumenterte mens de arbeidet med oppgaven på en datamaskin, ble oppgaven gjort om til en analog oppgave hvor jeg også inkluderte kjennetegnene til Bocconi et al. (2016). Det kan være kritikkverdig at informasjonsskrivet ikke ble endret, men på bakgrunn av NSD sin godkjennelse valgte jeg å bruke det samme informasjonsskrivet.

Selve studien

Det var 10 elever som deltok på studien min. Alle gikk i en 6.klasse på en Tromsøskole. Det var 6 gutter og 4 jenter som deltok. Når jeg fikk en ny sjanse til å gjennomføre studien valgte jeg å filme to og to elever i stedet for å filme alle ti samtidig. I tillegg ble algoritmene skrevet ut på papir. Filmingen gikk over to dager med en varighet på ca. 4 timer til sammen. Før jeg satte i gang filmingen hadde jeg et felles informasjonsmøte med elevene hvor jeg informerte om endringene i opplegget.

Elevene ble filmet parvis på et grupperom hvor en av elevene ble utstyrt med et GoPro-kamera. Oppgaven ble presentert på samme måte som i delkapittelet 3.4.1 Utforming av oppgaven. Hjelpemidler som elevene kunne ta i bruk var gradskive, penn og papir. Hvert elevpar fikk til sammen en klokketime på å løse oppgaven. Elevene ble opplyst om at jeg ikke kom til å skrive noen notater av dem underveis, men at jeg kom til å være tilstede mens filmingen pågikk. Jeg plasserte meg i bakgrunnen og sa at jeg var tilgjengelig for spørsmål dersom det var noe de lurte på. Jeg tok også vare på notatene som elevene skrev underveis.

3.5 Analysemetode

Hensikten med analyseprosessen er å gi leseren et innblikk i hvordan man som forsker har gått fram for å analysere datamaterialet sitt (Postholm et al., 2018, s. 139). Videre handler analyseprosessen ifølge Elo og Kyngäs (2008) om å:

a) Gjøre forskningen synlig og gjennomsiktig for leseren slik at slutningene fremstår som gyldige.
b) Ende opp med en samling av data som har lik mening og som svarer på forskningens problemstilling.
c) Bidra med ny kunnskap og ny innsikt på fenomenet som studeres.

I denne delen vil jeg synliggjøre disse tre punktene ved å vise hvordan jeg har gått fram for å sortere, redusere og strukturere datamaterialet til studiens endelig funn.

3.5.1 Induktiv og tematisk analyse

Problemstillingen for denne oppgaven er: *Hva kjennetegner elevers algoritmiske tenkning når de arbeider med geometri i blokkbasert programmering?* For å kunne svare på dette spørsmålet ønsket jeg å se på hvordan kjennetegnene *abstraksjon, generalisering* og *algoritmebehandling* kom til uttrykk gjennom hvordan elevene løste oppgaven.

Hvilken analysemetode man velger handler ifølge Elo og Kyngäs (2008) om hvorvidt forskeren har tilstrekkelig kunnskap og informasjon om fenomenet som skal studeres. Når forskeren ikke har tilstrekkelig kunnskap om fenomenet som skal studeres kan en induktiv analysemetode være hensiktsmessig å bruke. Da går man inn i datamaterialet med et «åpnere sinn». På bakgrunn av at algoritmisk tenkning fremstår som et noe unyansert begrep med ulike definisjoner og tolkninger, valgte jeg en induktiv og tematisk analyse. I en induktiv analyse tar man utgangspunkt i sitt eget datamateriale og utvikler koder og kategorier som beskriver fenomenet som skal studeres. Det betyr at funnene i forskningen er tettere knyttet opp mot den konteksten som studien ble utført i (Braun & Clarke, 2006, s. 12). I en tematisk analytisk tilnærming retter man oppmerksomheten mot temaet som undersøkes i studiet, det vil si at man ønsker å analysere temaet på tvers av datasettene. Ifølge Thagaard (2013, s. 183) krever en temasentrert analyse at alle informantene i studien får det samme spørsmålet eller den samme oppgaven. I min studie fikk alle de 10 elevene den samme oppgaven, dermed passet det å bruke en tematisk analyse på mitt datamateriale.

Analyseprosessen tar utgangspunkt i Braun og Clarke (2006) sine 6 faser for tematisk analyse: 1) Transkribering, 2) Generere koder, 3) Utvikle kategorier, 4) Gjennomgå kategoriene, 5) Definere og navngi kategoriene og 6) Skrive rapport.

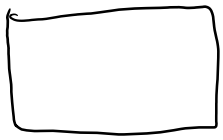
Fase 1: Transkribering

Fase 1 handler om skriftliggjøring av verbal og visuell data (Braun & Clarke, 2006, s. 17). Dette er den første fasen i møte med rådataen. I denne fasen anbefaler Braun og Clarke (2006) at man transkriberer og leser gjennom datamaterialet uten å begynne å kode.

Siden jeg hadde valgt videoopptak som metode for datainnsamling besto min transkripsjon av både verbal og visuell data. Jeg hadde 5 filmer på ca. 4 timer til sammen. Én film for hvert av elevparene. Jeg markerte korte pauser med …, og lange pauser med (…). Når elevene uttrykte seg verbalt skrev jeg det ned ordrett, mens når elevene gjorde bevegelser med kroppen skrev jeg det i parentes. Et eksempel på dette ser man under:

> **Elev 1**: Hmm … jeg tror jeg vet hva dette er! (peker på kvadratalgoritmen).

I tillegg la jeg inn illustrasjoner av hva elevene hadde tegnet:



Figur 7: Illustrasjon av et rektangel tegnet av en elev.

Når transkripsjonen var ferdig hadde jeg til sammen 5 dokumenter. Ett dokument for hvert av elevparene.

Fase 2: Generere koder i Nvivo

Fase 2 handler om å generere koder til datamaterialet ditt. En kode kan være en kommentar eller et notat på at noe i datamaterialet ditt virker interessant (Braun & Clarke, 2006, s. 18). I fase 2 valgte jeg å ta i bruk det datastyrte analyseverktøyet Nvivo.

Nvivo brukes innen kvalitativ forskning for å analysere såkalte ikke-numeriske data. Det vil si data som kommer i form av for eksempel tekst, lyd, bilde og video. Kort forklart hjelper Nvivo deg med å sortere og kombinere datamaterialet ditt gjennom å telle opp hvor mange ganger en kode eller en kategori opptrer i datamaterialet. Når man ønsker å kode et utsagn fra det transkriberte materialet, markerer man utsagnet og drar det over til de ulike kodene. På den måten kan man se hvor ofte en kode opptrer på tvers av datasettene.

Her er noen eksempler på hvordan jeg kodet noen av utsagnene:

Utsagn: *Det der er kvadratet! (peker på kvadratalgoritmen), fordi alle fire er jo 100, ikke sant?*

Kode: Beskriver likheter i steg-blokkene.

Utsagn: *Gå 80 steg (…) så det er nesten et like stort steg som i sta.*

Kode: estimerer 80 steg.

Utsagn: *Hva var et trapes? … Var ikke det liksom at de aldri møttes (strekker ut hendene).*

Kode: Forsøker å definere et trapes.

Utsagn: *Oi! klokka går den veien (lager en rund bevegelse med hånda). Vi skal gå motsatt vei av klokka. Vi må begynne på nytt igjen.*

Kode: Begynner forfra igjen.

Etter flere gjennomlesninger endte jeg opp med 77 koder som jeg mente var interessant for oppgavens problemstilling. Alle disse 77 kodene var skrevet inn i Nvivo. Her er et utdrag på noen av kodene.

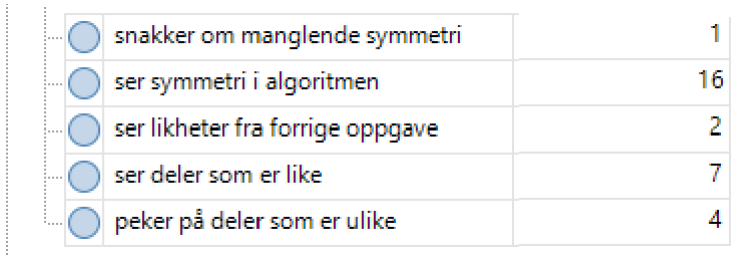
- Snakker om manglende symmetri.
- Ser symmetri i programmet.
- Ser likheter fra forrige oppgave.
- Ser deler som er like.
- Starter på nytt igjen.
- Oppfordrer til mer diskusjon.
- Representerer 100-steg blokken som 1 cm.
- Oppdager feil
- Tegner 10 prikker på arket

Fase 3 – Sortering av koder til innledende kategorier

Fase 3 begynner når hele datamaterialet er kodet og alle kodene er listet opp. Her begynner man å undersøke mønstre på tvers av datasettet (Braun & Clarke, 2006). I denne fasen brukes også begrepet analytisk koding. Dette er fordi kodingen er tettere knyttet opp mot problemstillingen for studien. På dette tidspunktet begynte jeg å se hvordan kjennetegnene abstraksjon, generalisering og algoritmebehandling kom til uttrykk i elevenes løsningsforslag.

Dermed var kodene i større grad «teoridrevet» enn «datadrevet» (Braun & Clarke, 2006). Ifølge Elo og Kyngäs (2008) er det en fordel med teoridrevne koder fordi man koder mer systematisk. På den andre siden kan man gå glipp av andre potensielle funn.

På dette tidspunktet hadde jeg 77 koder. For å sortere og redusere disse kodene begynte jeg å gå gjennom datamaterialet på nytt, nå flyttet jeg over utsagn, ord og kommentarer som passet inn i like koder. På den måten kunne kodene tallfestes.

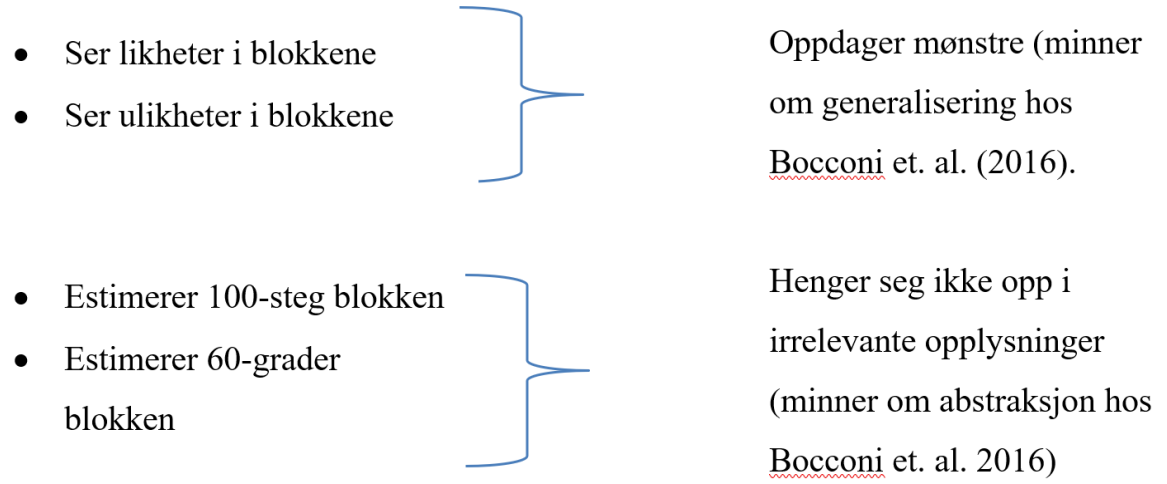


Figur 8: Koding i Nvivo.

Tallene i høyre kolonne viser hvor ofte ulike utsagn passet inn i kodene til venstre. Dette gjorde det lettere å ta bort koder som ikke var representativt og samtidig slå sammen koder som lignet på hverandre. På den måten begynte det å danne seg innledende kategorier.

Fase 4 - Gjennomgå kategoriene

Fase 4 begynner når man har et sett med kategorier. Nå begynner arbeidet med å revurdere om disse kategoriene faktisk er kategorier (Braun & Clarke, 2006, s. 91). I denne delen gikk jeg i større grad tilbake til teorikapittelet for å finne representative navn på kategoriene. Denne fasen kan knyttes til det Merriam (2009) kaller for teoretisering. Det vil si at man forsøker å definere og teoretisere kategoriene i tilknytning til tidligere forskning. Under ser man noen eksempler:



Fase 5 – Definere og navngi kategoriene

Fase 5 er den nest siste fasen. Her skal man definere de endelige kategoriene. Braun og Clarke skriver at det er viktig å gi kategoriene forståelig navn slik at de ikke virker for komplekse (Braun & Clarke, 2006).

Jeg utledet 2 kategorier for *abstraksjon,* 2 kategorier for *generalisering,* og 3 kategorier for *algoritmebehandling.* Abstraksjon: 1) Klarer å bruke ferdiglagde funksjonsblokker og 2) Klarer ikke å bruke ferdiglagde funksjonsblokker. Generalisering: 1) Oppdager mønstre i funksjonsblokkene, og 2) Oppdager ikke mønstre i funksjonsblokkene. Algoritmebehandling: 1) Følger algoritmene korrekt men vet ikke svaret, 2) Bruker algoritmene som «støtte» til tidligere kunnskap, og 3) Klarer ikke å følge algoritmene. Til sammen 7 kategorier.

Fase 6 – Presentere og analysere studiens funn

Sammenhengen mellom de 7 kategoriene resulterte i tre typer algoritmiske tenkere som jeg vil presentere i kapittel 4: Funn og analyse av funn.

3.6 Kvalitet i studien

Kvalitet i studien handler om studiens troverdighet og kan drøftes gjennom begrepene *validitet* og *reliabilitet.*

3.6.1 Validitet

Validitet er et mål på gyldighet og holdbarhet i forskning og handler om i hvilken grad en undersøkelse eller en studie måler det man ønsker å måle (Svartdal, 2009). Det vil si om studiens funn faktisk representerer den virkeligheten man har studert. Seale (1999, s. 38-40) skiller mellom indre og ytre validitet. Indre validitet handler om studiens funn er gyldige *innenfor* den konteksten den ble utført i, mens ytre validitet handler om studiens funn er gyldige *utenfor* den konteksten den ble utført i.

Indre validitet i min studie

Det som truer den indre validitet i min studie er at algoritmisk tenkning er et abstrakt begrep som det fortsatt er lite forskning på. Når det er en uoverensstemmelse om hva algoritmisk tenkning er, er det også vanskelig å si noe sikkert om hva som kjennetegner algoritmisk tenkning. Silverman (2011, s. 360-373) skriver at det som kan være med på å styrke den indre validiteten til en studie, er at man i analyseprosessen pendler mellom data og teori. På den måten unngår man å dra konklusjoner basert på enkle antakelser. Dette har jeg prøvd å oppnå gjennom å pendle mellom det Braun og Clarke (2006) kaller for «datadrevne» og «teoridrevne» koder. Allerede i fase 3 av analyseprosessen begynte jeg å utvikle innledende kategorier som kunne knyttes til kjennetegnene *abstraksjon, generalisering* og *algoritmebehandling* jf. Bocconi et al. (2016) sitt rammeverk. I teorikapittelet har jeg også vist at det er mange måter å tolke og forstå begrepet algoritmisk tenkning på, dermed har jeg også i min studie vært nødt til å gjøre noen valg. For eksempel valgte jeg å ikke se på kjennetegnene *feilsøking, dekomponering* eller *automatisering.* Hadde man inkludert disse kan det tenkes at også studiens funn hadde sett annerledes ut.

Et annet punkt som truer den indre validiteten er metodevalg. Med et kvalitativt og instrumentelt casestudie er det elevenes tolkninger av oppgaven som utgjør studiens empiri. Spørsmålet er da om videoopptakene av elevene representerer en virkelighet som jeg som forsker kan trekke slutninger om. Ifølge Postholm et al. (2018, s. 131) kan videoopptak virke forstyrrende på de som deltar i et forskningsprosjekt. Grunnen til dette er at videoopptak utleverer mange sider av de personene som deltar, noe som igjen kan bidra til at de oppfører seg annerledes enn hva de normalt ville gjort (Thagaard, 2013, s. 90).

Alle de 10 elevene som deltok i studien ble tatt ut fra en vanlig klasseromssituasjon og plassert i et grupperom med en medelev. I tillegg måtte en av elevene ha på et GoPro-kamera på hodet eller på brystet. Det kan godt tenkes at en slik kunstig læringssituasjon gjorde at elevene som deltok oppførte seg annerledes. Det er vanskelig å si i hvor stor grad bruk av video kan ha virket forstyrrende på elevene, men ved å kombinere videoopptak med en passiv observatørrolle var målet at elevene ikke skulle føle at de ble studert. Jeg forsøkte også å betrygge elevene om at det stort sett bare var hendene og oppgavearket som ble med på filmen.

Ytre validitet i min studie

Ved å benytte seg av en kvalitativ metode med et lite utvalg, vil studiens funn ikke kunne generaliseres til en større populasjon (Creswell, 2014, s. 203). Dette er naturligvis en svakhet ved å bruke en kvalitativ metode, men på den andre siden er målet med kvalitative metoder å få rike beskrivelse av det fenomenet eller av de personene som studeres. Postholm & Jacobsen (2018, s. 238) skriver at det derfor vil være mer naturlig å se på ytre validitet i form av ordet *overførbarhet.* En måte å måle overførbarhet på er gjennom å forsøke å plassere sin egen forskning inn i tidligere og ny forskning. På den måten viser man at sine egne analyser og tolkninger av datamaterialet gjøres innenfor en teoretisk ramme. Dette har jeg forsøk å oppnå gjennom å bruke Bocconi et al. (2016) sitt rammeverk. Allikevel vil det med bare 10 elever være vanskelig å si i hvilken grad studiens funn kan overføres til andre kontekster. Et større studie med flere informanter kunne nyansert og eventuelt styrket denne studiens funn.

3.6.2 Reliabilitet

Reliabilitet handler om å vise pålitelighet i forskningen (Silverman, 2011, s. 360-370). For at forskningen skal være reliabel må den være utført på en tillitsvekkende måte (Thagaard, 2013, s. 193). Ifølge Postholm et al. (2018, s. 224) er det to forhold som det kan være viktig å reflektere over når man snakker om reliabilitet i kvalitativ forskning. Det ene er refleksjon over egen subjektivitet, og det andre er gjennomsiktighet i forskningen.

Refleksjon over egen subjektivitet

For å styrke studiens reliabilitet er det viktig at man som forsker redegjør for hvordan man har opptrådd i forskningsfeltet (Thagaard, 2013, s. 194). Det vil si å avklare for leseren hvilke relasjoner man har hatt til de som har deltatt i studien. Gjennom å kombinere en fullstendig

observatørrolle med videoopptak har jeg forsøkt å holde avstand til elevene som deltok i studien. Dette var for å unngå å påvirke informantene og dermed unngå å påvirke datamaterialet.

Gjennomsiktighet i forskningsprosessen

Gjennomsiktighet i forskningen handler om at man som forsker må gjøre rede for hvordan data utvikles (Thagaard, 2013, s. 194). I analyseprosessen har jeg benyttet meg av en induktiv og tematisk analyse gjennom Braun og Clarke (2006) sine 6 faser. I tillegg er kodene utviklet til endelige kategorier gjennom det datastyrte programmet Nvivo. Jeg har dermed forsøkt å gjøre forskningen synlig og gjennomsiktig for leseren slik at de endelige kategoriene mine fremstår som troverdige. Gjennomsiktighet i forskningsprosessen vil også komme tydelig fram i kapittel 4: funn og analyse av funn. Her vil jeg vise hvordan elevene løste oppgaven gjennom å presentere utdrag, kommentarer og illustrasjoner fra det transkriberte datamaterialet.

3.7 Etiske retningslinjer

Prosjektet mitt var meldepliktig fordi studien innebærer behandling av personopplysninger. Prosjektet ble meldt inn til NSD og godkjent på bakgrunn av at det var i samsvar med personvernlovgivningen. Videre har jeg fulgt de etiske retningslinjene i henhold til « *De nasjonale forskningsetiske komiteer*» (NESH, 2016) gjennom tre prinsipper: *Informert samtykke, anonymitet* og *konsekvenser av å delta i forskningsprosjektet* (Thagaard, 2013, s. 26).

Informert samtykke

Felles for ethvert forskningsprosjekt er prinsippet om at man som forsker må ha deltakernes informerte og frie samtykke (Thagaard, 2013, s. 26). At samtykket er *fritt* handler om at deltakerne må godkjenne å delta på studien uten noe form for ytre press. At samtykket er *informert* handler om at deltakerne må vite hva det innebærer å delta.

Læreren som ga meg tilgang til å gjennomføre studien i hennes klasse informerte alle de 10 elevene om formålet og hensikten med prosjektet. Hun fikk godkjennelse av alle de 10 elevenes foresatte og av elevene selv. Samtykket ble gitt på bakgrunn av informasjonsskrivet som ble sendt med hjem til elevenes foresatte (vedlegg 2). Postholm et al. (2018, s. 249) skriver at en viktig del av det informerte samtykket er at deltakerne har forstått hva det er de samtykker til. For å forsikre meg om at elevene hadde forstått hva de hadde samtykket til, hadde jeg et informasjonsmøte med alle elevene hvor jeg på nytt gjennomgikk hensikten med studien og hvordan studien skulle gjennomføres.

Anonymitet

Anonymitet er det tredje prinsippet og handler om at man ikke skal kunne gjenkjenne deltakerne ut fra oppgaven (Thagaard, 2013, s. 26). For å ivareta elevenes anonymitet var kameraene vendt nedover mot bordet. Ved å bruke en tematisk analyse har jeg også nedtonet betydningen av hvordan hver elev løste oppgaven. Elevenes anonymitet er også bevart gjennom at navn er byttet ut med *Elev 1, Elev 2, Elev 3* osv. Elevenes dialekt er også byttet ut med bokmål.

Konsekvenser av å delta i forskningsprosjektet

Konsekvenser av å delta i forskningsprosjektet handler om at deltakerne i studien skal informeres om hva det innebærer å delta (Thagaard, 2013, s. 26). Elevene ble informert i informasjonsskrivet om hva studien gikk ut på og hva det ville innebære for dem å delta. I informasjonsskrivet sto det at det var frivillig å delta og at elevene når som helst kunne trekke seg fra prosjektet uten at det ville få noen negative konsekvenser for dem. Det sto også at de kom til å bli filmet med et GoPro-kamera og at de måtte være forberedt på å diskutere høyt. Elevene ble også gjort oppmerksom på endringer i opplegget med tanke på at oppgaven ikke skulle løses digitalt.

Kapittel 4

4 Funn og analyse av funn

Problemstillingen for denne oppgaven er: *Hva kjennetegner elevers algoritmiske tenkning når de arbeider med geometri i blokkbasert programmering?* Elevenes algoritmiske tenkning har blitt analysert induktivt og tematisk ut fra Bocconi et al. (2016) sine tre kjennetegn på algoritmisk tenkning: a*bstraksjon, generalisering* og *algoritmebehandling.*

4.1 Tre typer algoritmiske tenkere

Studiens funn viser tre typer algoritmiske tenkere: «*Den instrumentelle algoritmiske tenkeren», «Den prosedurale algoritmiske tenkeren»* og «*Den konseptuelle algoritmiske tenkeren»*. Bakgrunnen for disse er hvordan Bocconi et al. (2016) sine tre kjennetegn på algoritmisk tenkning kommer til uttrykk gjennom elevenes løsningsforslag.

For abstraksjon er det utviklet to kategorier med to nivåer: Nivå 1) Klarer å bruke funksjonsblokkene, og Nivå 0) Klarer ikke å bruke funksjonsblokkene. For generalisering er det utviklet to kategorier med to nivåer: Nivå 1) Oppdager mønstre i funksjonsblokkene, og Nivå 0) Oppdager ingen mønstre i funksjonsblokkene. For algoritmebehandling er det utviklet tre kategorier med tre nivåer: Nivå 2) Følger algoritmene korrekt med vet ikke svaret, Nivå 1) Bruker algoritmene som «støtte» til tidligere kunnskap, og Nivå 0) Klarer ikke å følge algoritmene.

Den instrumentelle algoritmiske tenkeren viser nivå 0 på abstraksjon, nivå 0 på generalisering og nivå 0 på algoritmebehandling. Den prosedurale algoritmiske tenkeren viser nivå 1 på abstraksjon, nivå 0 på generalisering og nivå 2 på algoritmebehandling. Den konseptuelle algoritmiske tenkeren viser nivå 1 på abstraksjon, nivå 1 på generalisering, og nivå 1 på algoritmebehandling. Sammenhengen mellom kategoriene og de tre algoritmiske tenkerne er presentert i tabellen på neste side.

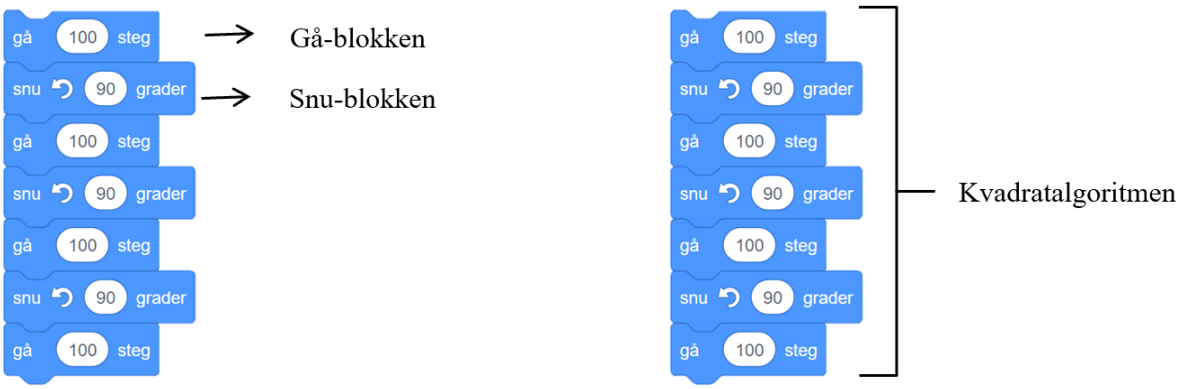
Tabell 2: Studiens funn: Tre typer algoritmiske tenkere.

Tre kjennetegn på algoritmisk tenkning:	**Kategorier:**	**Den instrumentelle algoritmiske tenkeren:** **(**4 elever)	**Den prosedurale algoritmiske tenkeren:** (2 elever)	**Den konseptuelle algoritmiske tenkeren:** (4 elever)
Abstraksjon (nivå 1)	Klarer å bruke ferdiglagde funksjonsblokker		X	X
Abstraksjon (nivå 0)	Klarer ikke å bruke ferdiglagde funksjonsblokker	X		
Generalisering (nivå 1)	Oppdager mønstre i funksjonsblokkene			X
Generalisering (nivå 0)	Oppdager ingen mønstre i funksjonsblokkene	X	X	
Algoritmebehandling (nivå 2)	Følger algoritmene korrekt men vet ikke svaret.		X	
Algoritmebehandling (nivå 1)	Bruker algoritmene som støtte til tidligere kunnskap.			X
Algoritmebehandling (nivå 0)	Klarer ikke å følge algoritmene.	X		

Tabellen er utviklet med utgangspunkt i hvordan de 7 kategoriene opptrer på tvers av de fem datasettene. Dermed vil de tre algoritmiske tenkerne inneholde noen av de samme kategoriene. Siden det er sammenhengen mellom disse kategorien som har resultert i de tre typene av algoritmiske tenkere, vil jeg i dette kapittelet analysere hver kategori. I diskusjonskapittelet vil jeg diskutere de tre algoritmiske tenkerne i lys av teori.

Begrepsavklaringer

For å lette lesingen i dette kapittelet vil jeg klargjøre noen begreper: Når det henvises til en samtale mellom to elever brukes betegnelsen: *Elev A* og *Elev B.* Når det henvises til generelle utsagn eller kommentarer som er tatt ut ifra ulike samtaler, brukes betegnelsen: *Elev 1, Elev 2, Elev 3* osv.



Gå-blokken og *snu-blokken* er to eksempler på to funksjonsblokker. Dersom det står: «*Elevene klarte å bruke funksjonsblokkene*», er det *gå-blokken og snu-blokken* det henvises til. *Algoritmen* viser til alle funksjonsblokkene som inngår i hele algoritmen, for eksempel *kvadratalgoritmen.* For å påminne leseren går oppgaven ut på at elevene skal identifisere firkanter som er skrevet i blokkbasert programmering. Oppgaven inneholder fem algoritmer for følgende firkanter: kvadrat, rektangel, rombe, parallellogram og trapes. Oppgaven er uten bruk av datamaskin og elevene samarbeider i par. I tillegg fikk elevene utdelt papir, penn og gradskive som hjelpemidler.

4.2 Abstraksjon

I følge Bocconi et al. (2016) handler abstraksjon i programmering om å se bort ifra irrelevant informasjon og fokusere på de riktige detaljene. Med dette mener Bocconi et al. at man må velge de riktige detaljene å fokusere på slik at problemet som skal løses blir mindre komplekst. I denne studien kom abstraksjon til uttrykk gjennom to kategorier som jeg valgte å kalle:

Nivå 1) Klarer å bruke ferdiglagde funksjonsblokker (6 elever)
Nivå 0) Klarer ikke å bruke ferdiglagde funksjonsblokker (4 elever)

Nivå 1) Klarer å bruke ferdiglagde funksjonsblokker (6 elever)

Seks elever klarte å bruke de ferdiglagde funksjonsblokkene. Det vil si at de valgte å fokusere på de riktige detaljene når de arbeidet med *gå-blokken* og *snu-blokken*. Både *gå-blokken* og *snu-blokken* er abstraksjoner av et linjestykke og en rotasjon/snuvinkel. Da er det viktigere å fokusere på hva disse blokkene betyr i en større sammenheng enn å bruke mye tid på å studere verdiene. Her er noen eksempler fra når disse elevene arbeidet med *gå-blokken:*

Elev 1: *Vi sier at dette er 100 steg (tegner opp en vilkårlig strek på papiret).*

Elev 2: *50 steg … så halvparten av 100 steg da (tegner en strek som er halvparten så lang som den andre).*

Elev 3: *Vi sier at dette er 80 steg (går et skritt som er litt kortere enn det forrige).*

Elev 4: *Dette er ca. 80 steg (tegner en strek som er nesten like lang som streken til 100 steg).*

Elev 5: *Hvis vi går 100 steg, sånn liksom (tar ett steg), så er 50 steg halvparten av det jeg gikk.*

Som man kan se bruker ikke disse elevene mye tid på å diskutere verdien i *gå-blokken,* men de oppfatter *gå-blokken* som enten et steg (Elev 3, Elev 5) eller som en vilkårlig strek på papiret (Elev 1, Elev 2, Elev 4). Dette sparer dem for mye tid i stedet for å begynne å måle

opp linjestykkene veldig nøyaktig. Slik jeg tolker det bruker disse elevene *gå-blokken* som en «mal» som de andre *gå-blokkene* kan justeres ut ifra. Elevene hadde en lignende tilnærming når de arbeidet med *snu-blokken*:

> ***Elev 1****: 180 grader ja […], hvis dette er 90 grader så må dette være 180 grader (eleven tegner opp en halvsirkel på papiret og markerer hvor 90 grader ville vært og dermed hvor 180 grader ville vært).*
>
> ***Elev 2****: Snu 120 grader … da må det være litt mer enn 90 grader (vrir seg litt mer enn 90 grader mot venstre).*
>
> ***Elev 3****: Så dette er ca. 60 grader .. okei, 120 grader.. okei, da må jeg snu meg dobbelt så mye.*

Hverken Elev 1, Elev 2 eller Elev 3 bruker en gradskive for å måle opp gradene, men prøver heller å justere seg ut ifra noen referansepunkt. Dette kan være en god strategi å bruke siden alle *snu-blokkene* er like på den måten at de går *mot* klokken.

Disse fire elevene klarte å bruke de ferdiglagde funksjonsblokkene på en god måte. De gjorde gode vurderinger av hva ordet *steg* kunne bety i denne sammenhengen, og de forsto at *snu-blokken* representerte en rotasjon i planet. Dette bidro til at de kunne arbeide seg effektivt nedover algoritmene fordi de brukte lite tid på å diskutere verdiene. Dette viser tegn til abstraksjon fordi de må «oversette» et blokkbasert programmeringsspråk til noen matematiske konsepter.

Nivå 0) Klarer ikke å bruke ferdiglagde funksjonsblokker (4 elever)

Fire elever klarte ikke å bruke de ferdiglagde funksjonsblokkene på en god måte. De brukte mye tid på å diskuterte verdiene i stedet for å fokusere på hva *gå-blokken* og *snu-blokken* representerte. Her er noen eksempler fra når elevene arbeidet med *gå-blokken:*

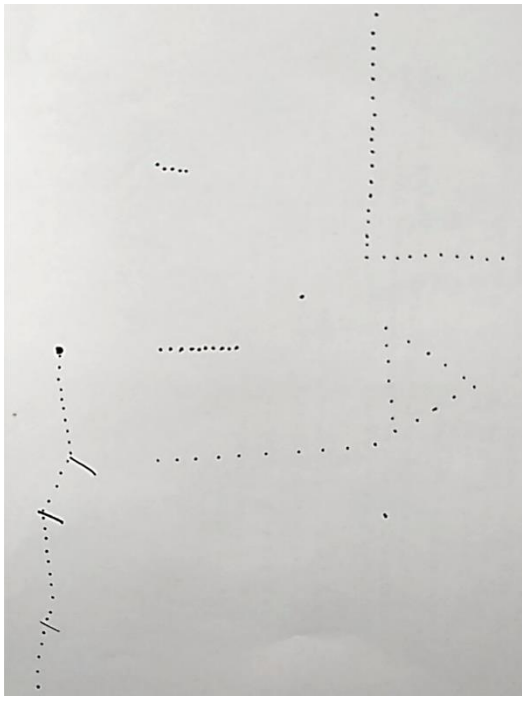
> ***Elev 1:*** *Hvordan skal vi vite hvor mye 100 steg er?*
>
> ***Elev 2:*** *100 steg … hvordan skal jeg ta 100 steg?*

Elev 3: *Hvordan skal man måle opp 100 steg?*

Elev 4: *Hvor store er 100 steg? Må jeg gå 100 skritt?*

Elev 5: *100 steg, skal du tegne 100 nå?*

Som man kan se er det utfordrende for disse elevene å forstå hva disse stegene representerer. Det er kanskje ikke så rart med tanke på at stegene representerer en verdi som det er vanskelig å si noe om (Haraldsrud et al., 2020, s. 95). I vanlig konstruksjon av geometriske figurer vil det være mer naturlig å arbeide med linjestykker som er oppgitt med en viss lengde, som for eksempel 5 cm. I Scratch operer man med steg, dermed kan tallet som står inni *gå-blokken* tolkes på mange ulike måter. Når de samme elevene arbeider med *snu-blokken* blir også dette en utfordring. I stedet for å se på *snu-blokken* som en rotasjon i planet, prøver de å konstruere vinkelen som en indre vinkel. Et eksempel på dette ser man i illustrasjonen under:



Figur 9: Elevenes oppgaveark. Bildet viser hvordan de har forsøkt å representere gå-blokken som prikker og hvordan de har rotert i planet.

Bildet over viser oppgavearket til to av elevene. Som man kan se har de valgt å representere *gå-blokken* som prikker. Man kan også se at det har vært utfordrende å rotere i planet, noe

som gjør at ingen av disse figurene ligner på firkanter. I Scratch opererer man med ytre vinkler i stedet for indre vinkler. Det vil si at når man skal snu seg 60 grader, så er det den ytre vinkelen som konstrueres. Den indre vinkelen blir dermed 180º- 60º = 120º. Dette var det flere av disse elevene som syntes var utfordrende. Dette viser også hvorfor det er viktig å fokusere på de riktige detaljene. Dersom disse elevene hadde fokusert på at det står *snu,* hadde det kanskje vært lettere å forstå at dette er en abstraksjon av en rotasjon.

Oppsummering av abstraksjon

Elevene som klarte å bruke de ferdiglagde funksjonsblokkene på en god måte arbeidet seg raskt fra blokk til blokk og fra algoritme til algoritme. De brukte verdiene i *gå-blokkene* og *snu-blokkene* som en «mal» som de kunne justere de andre blokkene etter. Dette viser et høyere abstraksjonsnivå fordi elevene klarte å se bort ifra irrelevante detaljer og heller fokusere på helheten. Elevene som ikke klarte å bruke de ferdiglagde funksjonsblokkene på en god måte, brukte mye tid på å diskutere verdiene i *gå-blokken* og *snu-blokken.* I stedet for å representere *gå-blokken* som et steg med beina eller som en strek på papiret, begynte de å tegne opp stegene som prikker. I tillegg forsøkte de å konstruere verdien i *snu-blokken* som indre vinkler i stedet for å rotere mot klokken. Dette viser et lavere abstraksjonsnivå fordi elevene tolker disse verdiene som noe veldig konkret i stedet for å se på selve funksjonen bak disse blokkene.

4.3 Generalisering

Generalisering handler ifølge Bocconi et al. (2016, s. 18) om å oppdage og identifisere mønstre og likheter i et datasett. Dermed er mønstergjenkjenning en viktig forløper til å kunne generalisere. I min studie kom generalisering til uttrykk gjennom to kategorier med to nivåer:

Nivå 1) Oppdager mønstre i funksjonsblokkene (4 elever)

Nivå 0) Oppdager ingen mønstre i funksjonsblokkene (6 elever)

Nivå 1) Oppdager mønstre i funksjonsblokkene (4 elever)

Fire elever oppdaget mønstre i funksjonsblokkene. Dette var tydeligst når de arbeidet med algoritmene for kvadratet, rektanglet og romben. Måten de oppdaget mønster på var gjennom å se på likheter og ulikheter i *gå-blokken* og *snu-blokken.* Her er noen eksempler fra når elevene oppdaget mønstre i *gå-blokken:*

Figur 10: Rombealgoritmen (t.v), kvadratalgoritmen (t.h).

> ***Elev 1:*** *Det der er kvadratet (peker på gå-blokken i rombealgoritmen) fordi alle fire er jo 100, ikke sant?.*

Elev 1 oppdager at *gå-blokken* gjentar seg fire gangen og at det er 100 steg i alle disse blokkene. Dermed tror denne eleven at rombealgoritmen må være kvadratet. Et kvadrat er definert som en firkant med fire like lange sider og fire rette vinkler (Van de Walle, Bay-Williams, Lovin & Karp, 2013, s. 268), dermed må også *snu-blokken* gjenta seg fire ganger og ha en verdi på 90 grader for at det skal kunne være et kvadrat. En rombe derimot er en firkant med fire like lange sider (Van de Walle et al., 2013, s. 268). Dette er det to elever som legger merke til:

> ***Elev 2****: Den der! Den den den! Det er en rombe (peker på rombealgoritmen). Fordi alle er 100!.*

> ***Elev 3:*** *Det er 100 steg på hver av de blokkene (peker på rombealgoritmen), så de er like lang, så det betyr at dette er en rombe.*

Elev 2 og Elev 3 klarer å identifiserer romben fordi de oppdager at 100 steg gjentar seg fire ganger i *gå-blokken*. Dermed må det bety at verdiene i *gå-blokkene* representerer noe som er

like langt. I denne oppgaven representerer de fire like lange linjestykker. En annen elev oppdager at rektangelalgoritmen har to *gå-blokker* som er like:

Figur 11: Rektangelalgoritmen

> ***Elev 4**: Hvis de går 100 steg og 100 steg, og de går 80 steg og 80 steg. Det er to sider som er like lange og to sider som også er like lange (peker på to og to gå blokker i rektangelalgoritmen) (…) det er et mønster med den der.*

Elev 4 påpeker at det er et mønster i rektangelalgoritmen ved at to og to *gå-blokker* har like verdier. Etter en liten stund sier eleven:

> ***Elev 4**: Okei, dette ER et rektangel (peker på rektangelalgoritmen). Vi gjentar jo det samme.*

Eleven kommenterer ikke verdiene i *snu-blokken* til rektangelalgoritmen. Et rektangel er en firkant med fire rette vinkler (Van de Walle et al., 2013, s. 268), noe de kunne ha oppdaget gjennom å se at *snu-blokken* gjentar seg med 90 grader.

Den andre måten disse seks elevene oppdaget mønstre i funksjonsblokkene på var gjennom å oppdage ulikheter i *gå-blokken* og *snu-blokken.* Her er et eksempel fra når tre elever sammenligner rombealgoritmen og kvadratalgoritmen:

Figur 12: Rombealgoritmen (t.v), Kvadratalgoritmen (t.h).

> ***Elev 1**: Det der er en rombe (peker på rombealgoritmen). Det vet jeg, fordi den har 100,100, 100, 100. Da må det der være et kvadrat (peker på kvadratalgoritmen), fordi den har 100, 100, 100, 100 og 90 grader hele veien.*
>
> ***Elev 2**: Det der må være en rombe (peker på rombealgoritmen) og det der må være et kvadrat (peker på kvadratalgoritmen), fordi kvadratet har 90 grader hele veien, mens den (peker på rombealgoritmen) har ikke det.*
>
> ***Elev 3:** Hvis disse ikke er like som resten (peker på 80-steg blokken på rektangelalgoritmen), så er det ikke et kvadrat, det er et rektangel.*

Som man kan se på figuren over ligner algoritmene på hverandre, men det som skiller seg ut er at kvadratet har en 90-grader *gå-blokk* som gjentar seg. Dette oppdager disse elevene. En annen elev studerer algoritmen for romben og algoritmen for parallellogrammet og sier:

> ***Elev 4:*** *Okei, jeg har funne ut noe sykt. En rombe er et parallellogram, bare at et parallellogram ikke har fire like lange sider. Romben har 100, 100,100,100, mens parallellogrammet har 100,50,100,50.*

Jeg tolker det som om at denne eleven vet at en rombe også er et trapes fordi begge firkantene har to par parallelle sidekanter. Dermed klarer eleven å identifisere to firkanter samtidig gjennom å se på ulikheter i *gå-blokken.*

Mønstergjenkjenning er en viktig forløper til det å generalisere, hvor det å generalisere handler om å trekke ut en egenskap som er felles for alle klasser av et tilfelle (Davis et al., 2011). To av disse fire elevene oppdaget hva som var felles for alle de fem algoritmene:

> ***Elev 1****: Åh! Alle har jo fire kanter! (drar fingeren over alle algoritmene)*

> ***Elev 2****: Alle er like lange fordi de har fire kanter […] For en femkant ville det vært to blokker til, en sekskant to blokker til (bruker to fingre og peker på tomrommet under algoritmene).*

Dette er et steg opp fra å bare oppdage mønstre mellom de ulike funksjonsblokkene. Disse eleven oppdager mønstre i alle algoritmene. På grunn av at alle algoritmene representerer firkanter, vil de ha det til felles at *gå-blokken* gjentar seg fire gange. Dette var det allikevel

bare to elever som sa noe om. Elev 2 kommenterer i tillegg hvordan algoritmene kan utvikles videre for at det skal bli en femkant eller en sekskant.

Nivå 0) Oppdager ingen mønstre i funksjonsblokkene (6 elever)

De seks resterende elevene oppdaget ingen mønstre i funksjonsblokkene. Ingen av disse elevene kommenterte hvordan for eksempel verdiene i *gå-blokkene* eller *snu-blokkene* gjentok seg i flere av disse algoritmene. Dermed har jeg ingen eksempler fra datamaterialet å vise her. Men siden denne kategorien henger sammen med de andre kategoriene, er det viktig å påpeke at også dette er et funn.

Oppsummering av generalisering

Elevene som oppdaget mønstre i funksjonsblokkene klarte å identifisere kvadratet, romben, rektangelet og parallellogrammet. De så hvordan *gå-blokken* og *snu-blokken* gjentok seg på ulike måter i algoritmene og identifiserte firkantene på bakgrunn av dette. Elevene som ikke oppdaget mønstre i funksjonsblokkene kommenterte ingenting om likheter og ulikheter i *gå-blokken* og *snu-blokken.*

4.4 Algoritmebehandling

Algoritmebehandling handler om hvordan man klarer å følge og forklare et sett med instrukser (Bocconi et al. 2016). I denne studien kom algoritmebehandling til uttrykk gjennom tre kategorier med tre nivåer:

1) Følger algoritmene korrekt men vet ikke svaret (2 elever)

2) Bruker algoritmene som «støtte» til tidligere kunnskap (4 elever)

3) Klarer ikke å følge algoritmene (4 elever).

Nivå 2) Følger algoritmene korrekt men vet ikke svaret (2 elever)

To elever klarte å følge algoritmene korrekt men de klarte ikke å koble svaret til problemet som skulle løses. Det vil si at de klarte å lage like kopier som de man ville fått i Scratch dersom man hadde kjørt programmet. Problemet var at de ikke visste hvilke figurer de hadde endt opp med. Her er et eksempel fra når de arbeidet med parallellogramalgoritmen:

Figur 13: Parallellogramalgoritmen

Elev A: *Du må si hva jeg skal gjøre (den ene eleven stiller seg opp på gulvet).*

Elev B: *Gå 1 meter, også snur du deg 60 grader (eleven snur seg mot venstre).*

Elev A: *Etter 60 grader skal jeg?*

Elev B: *Da skal du gå 50 steg.*

Elev B: *Så snur du deg 120 grader.*

Elev A: *90 grader er sånn ca … så da er 120 litt mer (snur seg mot venstre).*

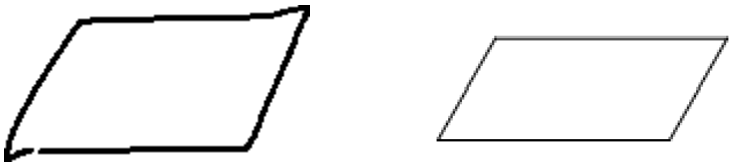
Elev B: *Ja, hvis du snur deg helt til venstre snur du deg 180 grader, så da må 120 grader være … (går bort å snur den andre eleven dit han tror 120 grader må være).*

Elev B: *Så går du 100 steg og snur deg 60 grader også 50 steg.*

Elev A: *Og det var det?.*

Elev B: *Ja.*

Resultatet blir følgende:



Figur 14: En elev følger instruksjoner (figur t.v), figuren man ville fått i Scratch (figur t.h).

Som man kan se klarer Elev A å følge algoritmene på en korrekt og bestemt måte. Det er også tydelig i måten elevene kommuniserer på. Tydelige instruksjoner gir gode resultater. Dermed ender de opp med en figur som ville vært veldig lik den man hadde fått i Scratch dersom man hadde kjørt algoritmen på datamaskinen. Problemet var at disse to elevene ikke visste hvilken figur dette var. Det samme problemet skjedde når de arbeidet med rombealgoritmen:

Figur 15: Rombealgoritmen

Elev B: *Gå 100 steg.*

Elev A*: Ja! så var det 60 grader var det ikke?.*

Elev B*: 60 grader ja! Så går du 100 steg.*

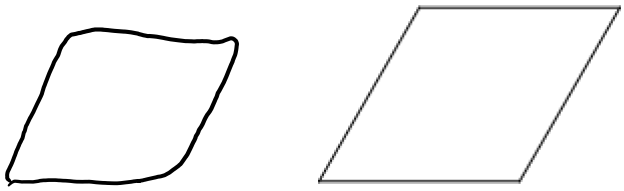
Elev A*: 100 steg (sier steget høyt og går ett skritt).*

***Elev B**: 120 grader.*

***Elev A**: 120 grader (sier steget høyt og går ett skritt).*

***Elev B:** 60 grader også 100 steg igjen.*

Resultatet blir følgende:



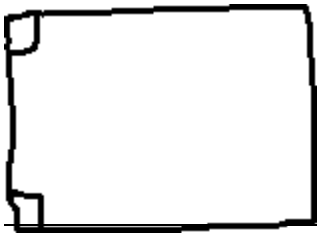
Figur 16: En elev følger instruksjoner (figur t.v), figuren man ville fått i Scratch (figur t.h).

Også denne gangen klarer elevene å konstruere en lik firkant som den man ville fått i Scratch. Problemet er at også denne gangen vet ikke elevene hvilken figur dette er. Allikevel klarte disse to elevene å følge alle de fem algoritmene korrekt. De gjorde ingen slurvefeil underveis og gjorde gode tolkninger av funksjonsblokkene.

Nivå 1) Bruker algoritmene som en «støtte» til tidligere kunnskap (4 elever)

Fire elever brukte algoritmene som en «støtte» til tidligere kunnskap. Det vil si at de ikke var så opptatte av å følge algoritmene veldig nøyaktig, men lagde heller raske tegninger for å illustrere et poeng. Her er et eksempel på to elever som arbeider med kvadratalgoritmen:

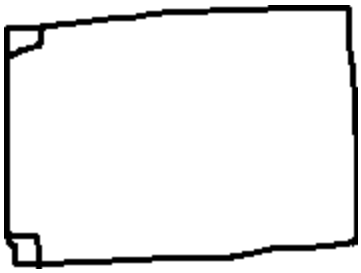
***Elev 1:** Vi kan begynne med å finne ut hvem kvadratet er, det er ganske enkelt. For det er liksom 100 steg, 90 grader, 100 steg (peker på kvadratalgoritmen).For det er liksom (tegner en rask skisse). Sånn, kvadratet.*



Figur 17: En elev tegner opp kvadratalgoritmen.

Eleven tegner bare en rask skisse for å illustrere poenget sitt. Eleven stopper ikke opp for å vurdere hver blokk. Den samme framgangsmåten kunne man finne hos to andre elever.

> ***Elev A:*** *Ja det blir vel mer som, der, 90 grader sånn, også 80 steg sånn der, også enda 90 grader ja, også 100, også opp igjen (lager en rask skisse).*
>
> ***Elev B:*** *Ja. Det blir rektanglet.*



Figur 18: En elev tegner opp rektangelalgoritmen.

I likhet med de andre to elevene er heller ikke disse elevene opptatt av å gjøre en grundig analyse av hver blokk. Det kan virke som om at disse elevene på forhånd vet hva både et kvadrat og et rektangel er. Egenskapene til disse figurene gjenkjennes i funksjonsblokkene, noe som gjør at algoritmene bare blir en ekstra støtte til noe kjenner fra før. Utfordringen med denne strategien var at den bare fungerte på algoritmene for kvadratet, romben og rektanglet. Når de arbeidet med algoritmen for parallellogrammet og trapeset, justerte de tegningene sine ut ifra hva hvordan de trodde at figurene skulle se ut. Ved å ignorere viktige detaljer i funksjonsblokkene var det flere av disse elevene som ikke klarte å identifisere parallellogrammet og trapeset.

Nivå 0) Klarer ikke å følge algoritmene (4 elever).

Fire av elevene klarte ikke å følge algoritmene. De gjorde slurvefeil underveis som også gjorde at hele figuren ble feil. Her er et eksempel på når to av disse elevene arbeidet med parallellogramalgoritmen.

Elev A: *Vi begynner med å gå 100 steg. 1,2,3,4,5…*

Elev B: *Skal du tegne 100 steg nå?*

Elev A: *Sånn, 100 steg er 10 prikker.*

Elev A*: Så snur vi oss 60 grader, høyre eller venstre?*

Elev B*: Vi skal venstre… jeg er ikke så godt på sånt.*

Elev A: *Da tar vi høyre … også 50 skritt. 1,2,3,4,5 (teller høyt og tegner opp prikker).*

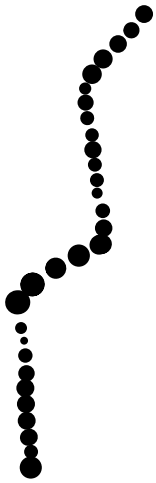
Elev B*: Hmm, jeg tror jeg begynner å få en liten anelse på hva det kan være.*

Elev A: *Jeg har ikke peiling jeg.*

Elev B: *Så skal vi snu 180 grader (det står 120 grader, men ingen oppdager dette).*

Elev A: *Dette gir ingen mening. Okei, 60 grader. Det må være her, sånn. Også 50 skritt.*

Her er det tydelig at elevene ikke er så nøye i hvordan de følger blokkene. En liten slurvefeil underveis gjør at også resten blir feil. For eksempel velger Elev A å snu til høyre selv om Elev B sier de skal snu til venstre. Elevene ender opp med en figur som er helt ulik den vi ville fått i Scratch.



Figur 19: Illustrasjon av hvordan en elev prøver å følge parallellogramalgoritmen.

Det som er litt overraskende er at disse to elevene ikke stopper opp underveis for å vurdere tegningen underveis. På forhånd vet de at alle algoritmene representerer firkanter, dermed burde de også stoppe opp for å dobbeltsjekke sine egne tolkninger av blokkene. Dette var noe som kjennetegnet alle de fire elevene som havnet i denne kategorien.

Oppsummering av algoritmebehandling

To elever klarte å følge algoritmene korrekt og nøyaktig, men visste ikke hvilken figur de hadde tegnet. Fire elever brukte algoritmene som «støtte» til tidligere kunnskap og var ikke like opptatt av å følge hver blokk nøyaktig. Dette fungerte godt når elevene arbeidet med algoritmene for kvadratet, romben og rektanglet, men fungerte dårligere når elevene ignorerte

verdiene i funksjonsblokkene og justerte tegningene sine ut ifra hva de trodde algoritmene kom til å bli. Fire elever klarte ikke å følge algoritmene og gjorde mange slurvefeil underveis.

Oppsummering av alle kategoriene:

Oppsummert ser man hvordan de tre kjennetegnene til Bocconi et al. (2016) kom til uttrykk gjennom hvordan elevene løste oppgaven:

Abstraksjon: to kategorier med to nivåer.

Nivå 1) Klarer å bruke ferdiglagde funksjonsblokker (6 elever)
Nivå 0) Klarer ikke å bruke ferdiglagde funksjonsblokker (4 elever)

Generalisering: to kategorier med to nivåer.

Nivå 1) Oppdager mønstre i funksjonsblokkene (4 elever).

Nivå 0) Oppdager ingen mønstre i funksjonsblokkene (6 elever).

Algoritmebehandling: tre kategorier med tre nivåer.

2) Følger algoritmene korrekt men vet ikke svaret (2 elever)

1) Bruker algoritmene som «støtte» til tidligere kunnskap (4 elever)

3) Klarer ikke å følge algoritmene (4 elever).

Kapittel 5

5 Diskusjon

Problemstillingen for denne masteroppgaven er: *Hva kjennetegner elevers algoritmiske tenkning når de arbeider med geometri i blokkprogrammering?* Studiens funn viser tre typer algoritmiske tenkere: «*Den instrumentelle algoritmiske tenkeren», «Den prosedurale algoritmiske tenkeren»* og «*Den konseptuelle algoritmiske tenkeren»*. Disse er utviklet på bakgrunn av hvordan de 7 kategoriene for abstraksjon, generalisering og algoritmebehandling kommer til uttrykk i elevenes løsningsforslag.

I denne delen vil jeg diskutere hvorfor elevene har blitt delt inn i tre typer algoritmiske tenkere og hvorfor de viser ulike kjennetegn på algoritmisk tenkning. I del 5.4 vil jeg diskutere sammenhengen mellom disse tre og forsøke å svare på hva som kjennetegner algoritmisk tenkning.

5.1 Den instrumentelle algoritmiske tenkeren

Fire elever har fått betegnelsen som «Den instrumentelle algoritmiske tenkeren», og består av følgende tre kategorier:

- Klarer ikke å bruke ferdiglagde funksjonsblokker (Abstraksjon nivå 0).
- Oppdager ingen mønstre i funksjonsblokkene (Generalisering nivå 0).
- Klarer ikke å følge algoritmene (Algoritmebehandling nivå 0).

Den instrumentelle algoritmiske tenkeren klarer ikke å bruke de ferdiglagde funksjonsblokkene på en god måte og bruker mye tid på å tolke verdiene i *gå-blokken* og *snu-blokken.* Verdiene i *gå-blokken* uttrykkes som prikker hvor 100 steg blir omgjort til 10 prikker, 80 steg som 8 prikker og 50 steg som 5 prikker. Verdiene i *snu-blokken* blir tolket som en indre vinkel som elevene prøver å konstruere med en gradskive. Ifølge Bocconi et al. (2016) handler abstraksjon i programmering om å se bort ifra irrelevante detaljer og fokusere på det som er viktigst. Det kan tyde på at elevene synes det er vanskelig å forstå at *gå-blokken* representerer et linjestykke, mens *snu-blokken* representerer en rotasjon i planet. I stedet for å fokusere på funksjonene bak disse blokkene, henger de seg opp i hva verdiene kan bety. På

bakgrunn av at disse elevene henger seg opp i irrelevante detaljer, har jeg plassert den instrumentelle algoritmiske tenkeren på nivå 0 under kjennetegnet *abstraksjon*.

Når det kommer til generalisering oppdager ikke den instrumentelle algoritmiske tenkeren noen mønstre i funksjonsblokkene. Ingen av disse elevene kommenterer hvordan *gå-blokken* eller *snu-blokken* gjentar seg i flere av algoritmene. En viktig forløper til generalisering er ifølge Bocconi et al. (2016) å identifisere likheter og ulikheter i et datasett. Dermed vil det være lettere å løse nye problemer som ligner på hverandre. En av grunnene til at jeg valgte å bruke fem firkanter i oppgaven som elevene fikk, var for å synliggjøre hvordan et mønster kunne se ut i blokkbasert programmering. Grunnen til dette er fordi alle algoritmene har en *gå-blokk* som gjentar seg fire ganger, dermed må det også bety at alle algoritmene er firkanter. Siden ingen av disse elevene kommenterer mønstrene i funksjonsblokkene har jeg plassert den instrumentelle algoritmiske tenkeren på nivå 0 under kjennetegnet *generalisering*.

Når det kommer til algoritmebehandling er det vanskelig for den instrumentelle algoritmiske tenkeren å følge algoritmene på en god måte. Noen små slurvefeil underveis gir systematiske feil og elevene ender opp med helt andre figurer enn de man ville fått i Scratch. En viktig del av algoritmebehandling er ifølge Bocconi et al. (2016) å kunne vurdere algoritmer. Dermed må algoritmen være skrevet på en stegvis og korrekt måte for at datamaskinen skal kunne følge den. Siden disse elevene ikke klarer å følge algoritmene på en nøyaktig måte har jeg plassert dem på nivå 0 under kjennetegnet *algoritmebehandling.*

Denne instrumentelle algoritmiske tenkeren er som navnet tilsier, inspirert av Skemp (1976) sitt begrep om instrumentell forståelse i matematikk. Under delkapittelet 2.2.3 står det at algoritmisk tenkning *kan* forstås i lys av instrumentell forståelse fordi bruk av algoritmer i skolesammenheng har møtt kritikk for å ikke fremme matematisk forståelse. På samme måte kan man i min studie se en lignende tilnærming til programmering. Den instrumentelle algoritmiske tenkeren velger ut en algoritme nokså tilfeldig og jobber seg nedover hver blokk uten å vite om algoritmen fungerer. Videre kan man spørre hvorfor den instrumentelle algoritmiske tenkeren havner på nivå 0 på alle kjennetegnene? For å kunne svare på dette må jeg trekke inn noe ny teori. Haraldsrud et al. (2020, s. 102) skriver om det som i pedagogisk og didaktisk litteratur kalles for «cognitive load». Dette kan oversettes til hjernekapasitet eller

minnekapasitet. I programmeringssammeneheng handler dette om at mye av elevers hjernekapasitet går til å tolke syntaksen til programmeringsspråket som brukes. Dermed vil elever først og fremst bruke mye energi på å forstå hva kodene betyr. Dette vil igjen gå på bekostning av det faglige aspektet. Min tolkning er at den instrumentelle algoritmiske tenkeren får en såkalt cognitive load som igjen kan knyttes til lav grad av abstraksjon. Når elevene bruker mye tid på å diskutere detaljer har de ikke overskudd eller overblikk til å oppdage mønstre i funksjonsblokkene eller vurdere algoritmene underveis. Dette blir tydelig i hvordan den instrumentelle algoritmiske tenkeren jobber. Mye av konsentrasjonen går til å tolke funksjonsblokkene, noe som igjen gjør det vanskelig å vurdere algoritmene underveis.

5.2 Den prosedurale algoritmiske tenkeren

To elever har fått betegnelsen som «Den prosedurale algoritmiske tenkeren», og består av følgende tre kategorier:

- Klarer å bruke ferdiglagde funksjonsblokker (Abstraksjonsnivå 1).
- Oppdager mønstre i funksjonsblokkene (Generalisering nivå 0)
- Følger blokkene korrekt med vet ikke svaret (Algoritmebehandling 2).

Den prosedurale algoritmiske tenkeren klarer å bruke funksjonsblokkene på en god måte. Dette gjør de ved å representerer verdiene i *gå-blokkene* og *snu-blokkene* på en fornuftig måte uten å henge seg opp i detaljer. Disse elevene velger å instruere den andre til å bevege seg etter blokkene, noe som tyder på at de tolker *snu-blokken* som en ytre vinkel i stedet for en indre vinkel. Abstraksjon er ifølge Bocconi et al. (2016) det viktigste kjennetegnet på algoritmisk tenkning fordi alle programmeringsspråk er forenklinger av virkeligheter. Siden disse to elevene klarer å forstå at de to funksjonsblokkene er forenklinger av et linjestykke og en ytre vinkel, har jeg plassert dem på nivå 1 under kjennetegnet *abstraksjon.*

Når det kommer til generalisering oppdager heller ikke den prosedurale algoritmiske tenkeren noe mønstre i funksjonsblokkene. Dermed plasseres også den prosedurale algoritmiske tenkeren på nivå 0 under kjennetegnet *generalisering.*

Når det kommer til algoritmebehandling klarer den prosedurale algoritmiske tenkeren å følge algoritmene korrekt. I likhet med den instrumentelle algoritmiske tenkeren velges algoritmene ut tilfeldig, men den prosedurale algoritmiske tenkeren klarer å tolke funksjonsblokkene på en

god måte. Siden en viktig del av algoritmebehandling handler om klare å følge algoritmer på en stegvis og bestemt måte (Bocconi et al., 2016), har jeg plassert den prosedurale algoritmiske tenkeren på nivå 2 under kjennetegnet *algoritmebehandling*.

Den prosedurale algoritmiske tenkeren er som navnet tilsier, inspirert av Hiebert og Lefevre (1986) sitt begrep om prosedural forståelse i matematikk. Under delkapittelet 2.2.3 står det at algoritmisk tenkning *kan* forstås i lys av prosedural forståelse fordi algoritmer i matematikk er abstraksjoner av den virkelige verden (Brousseau, 2006). Det vil si at elever som behersker et mer formelt matematisk språk også i større grad behersker bruk av algoritmer. Dermed vil det også være naturlig å snakke om prosedural forståelse i programmering fordi elever må klare å kommunisere til en datamaskin hva den skal gjøre på et språk som datamaskinen forstår (Papert, referert i Grover & Pea, 2013).

Både matematikkfaget og programmeringsfaget har det til felles at de bruker sine egne notasjonssystem for å forenkle virkeligheten, men de opererer med egne symboler og egne regler. Den prosedurale algoritmiske tenkeren viser i min studie et høyere abstraksjonsnivå fordi disse elevene klarer å «oversette» et blokkprogrammeringsspråk til et matematisk språk (abstraksjon nivå 1). Videre skriver Hiebert og Lefevre (1986, s. 6) at prosedural forståelse i matematikk handler om at man klarer å følge instruksjoner for å komme fram til et rett svar, det vil si at man klarer å følge et sett med instruksjoner på en bestemt og tydelig måte. Den prosedurale algoritmiske tenkeren viser i min studie at de klarer å følge de fem algoritmene presist og nøyaktig (algoritmebehandling 2). Videre kan man spørre seg hva grunnen er til at den prosedurale algoritmiske tenkeren kommer på et høyere nivå på abstraksjon og algoritmebehandling? Abstraksjon henger sammen med algoritmebehandling fordi en algoritme er en abstraksjon av virkeligheten (Bocconi et al., 2016; Brousseau, 2006). Når disse elevene ikke trenger å bruke mye tid på å forstå hva funksjonsblokkene betyr, er det naturlig å tenke at dette bidrar til å frigjøre elevenes hjernekapasitet (Haraldsrud et al., 2020). De kan heller konsentrere seg om å følge algoritmene og klarer dermed å arbeide seg systematisk nedover alle blokkene. Utfordringen er derimot at de ikke klarer er å koble svaret de får til problemet som skal løses. Det faglige aspektet ved oppgaven uteblir i elevenes diskusjoner noe som også kan henge sammen med at de ikke oppdager mønstre i funksjonsblokkene (generalisering 0). .

5.3 Den konseptuelle algoritmiske tenkeren

Fire elever fikk betegnelsen som «Den konseptuelle algoritmiske tenkeren kjennetegnes», og består av følgende tre kategorier:

- Klarer å bruke funksjonsblokkene (Abstraksjon nivå 1).
- Oppdager mønstre i funksjonsblokkene (generalisering nivå 1).
- Bruker algoritmene som «støtte» til tidligere kunnskap (Algoritmebehandling 1).

Den konseptuelle algoritmiske tenkeren klarer å bruke funksjonsblokkene gjennom å estimere og anslå verdiene i *gå-blokken* og *snu-blokken.* Siden disse elevene klarer å forstå at de to funksjonsblokkene er forenklinger av et linjestykke og en ytre vinkel, har jeg i likhet med den prosedurale algoritmiske tenkeren plassert dem på nivå 1 under kjennetegnet *abstraksjon.*

Den konseptuelle algoritmiske tenkeren oppdager mønstre i funksjonsblokkene, og oppdager likheter og ulikheter i både *gå-blokken* og *snu-blokken.* Mønstergjenkjenning er igjen en forløper til generalisering fordi en viktig del av å programmere er å utvikle generelle algoritmer som kan løse et sett med like problemer (Bocconi et al., 2016). Alle disse elevene oppdager også generelle likhetstrekk på tvers av alle algoritmene fordi *gå-blokken* gjentar seg fire ganger. To av disse fire elevene kommenterer i tillegg hvordan algoritmene kan utvikles slik at de blir femkanter og sekskanter. Dette viser en utvikling mot en mer generell tankegang, som vil være en viktig del av å programmere (Bocconi et al., 2016). På bakgrunn av dette har jeg plassert den konseptuelle algoritmiske tenkeren på nivå 1 under kjennetegnet *generalisering.*

Den konseptuelle algoritmiske tenkeren bruker algoritmene som en «støtte» til tidligere kunnskap. I stedet for å velge ut en algoritme litt tilfeldig, begynner elevene med å diskutere egenskapene til de ulike firkantene. Deretter begynte de å lete etter algoritmen som passer til egenskapene. En viktig del av algoritmebehandling er å kunne følge algoritmene nøyaktig (Bocconi et al., 2016). Siden den konseptuelle algoritmiske tenkeren bruker algoritmene som «støtte» til tidligere kunnskap ignorerer de ofte viktige detaljer i blokkene. Dermed justerer de sine egne tegninger ut ifra hva de tror det kommer til å bli. På bakgrunn av dette har jeg plassert den prosedurale algoritmiske tenkeren på nivå 1 under kjennetegnet *algoritmebehandling.*

Den konseptuelle algoritmiske tenkeren er som navnet tilsier, inspirert av Hiebert og Lefevre (1986) sitt begrep om konseptuell forståelse i matematikk, men det også mulig å bruke også Skemp (1976) sitt begrep om relasjonell forståelse. Jeg har allikevel valgt å bruke betegnelsen konseptuell. Under delkapittelet 2.2.3 står det at algoritmisk tenkning *kan* forstås i lys av konseptuell forståelse fordi ordet *tenkning* viser til at algoritmisk tenkning må bety noe mer enn å bare anvende standardalgoritmer i matematikk. Det må også handle om å forstå hvorfor en algoritme fungerer. Den konseptuelle algoritmiske tenkeren er den eneste som oppdager mønstre i funksjonsblokkene (generalisering nivå 1), noe som også kan henge sammen med abstraksjon (abstraksjon nivå 1). Grunnen til dette kan være at man i matematikk generaliserer over abstrakte objekter (Davis et al., 2011, s. 150). I denne oppgaven er det de to funksjonsblokkene som er de abstrakte objektene. Men de er også en abstraksjon av to geometriske konsepter. Selv om den konseptuelle algoritmiske tenkeren ikke klarer å følge algoritmene på en bestemt og nøyaktig måte (algoritmebehandling nivå 1), er det blant disse fire elevene at matematikken kommer fram i elevenes diskusjoner. Dette kan også knyttes til det Haraldsrud et al. (2020) kaller for minnebelastning (cognitive load). Den konseptuelle algoritmiske tenkeren bruker lite tid på å følge algoritmene nøyaktig, men fokuserer heller på å diskutere det matematiske problemet som skal løses.

5.4 Hva kjennetegner algoritmisk tenkning?

Sammenhengen mellom de 7 kategoriene har resultert i tre typer algoritmiske tenkere. Spørsmålet er hvem av disse tre som viser flest kjennetegn på algoritmisk tenkning? For å lette lesingen vil jeg i denne delen bruke betegnelsen INS.AT for den instrumentelle algoritmiske tenkeren, PRO.AT for den prosedurale algoritmiske tenkeren, og KON.AT for den konseptuelle algoritmiske tenkeren.

Den INS.AT og den PRO.AT har flere likhetstrekk fordi ingen oppdager mønstre i funksjonsblokkene (generalisering nivå 0). I tillegg har de samme strategi for å «angripe» problemet. Begge velger en algoritme tilfeldig og starter på den øverste blokkene for så å jobbe seg nedover hele algoritmen. Dermed kan både den IN.AT og den PR.AT minne om det Skemp (1976) kaller for en instrumentell forståelse i matematikk. Det som skiller den PRO.AT fra den INS.AT, er det Hiebert og Lefevre (1986) kaller for *notasjon*. Elever med en prosedural forståelse i matematikk behersker også det matematiske språket. I dette tilfellet

behersker den PRO.AT blokkprogrammeringsspråket og viser dermed et høyere abstraksjonsnivå enn den IN.AT (Abstraksjon nivå 1).

Den PRO.AT og den KON.AT klarer begge å håndtere funksjonsblokkene på en god måte (Abstraksjon nivå 1), men de skiller seg ut på det at det kun er den KON.AT som oppdager mønstre i funksjonsblokkene (generalisering nivå 1). I tillegg har de to ulike måter å bruke algoritmene på. Den PRO.AT klarer å følge blokkene bestemt og tydelig, men klarer ikke å koble svaret til problemet som skal løses (Algoritmebehandling nivå 2). Den KON.AT bruker algoritmene som en «støtte» til tidligere kunnskap og diskuterer i større grad det matematiske problemet som skal løses. Den KON.AT klarer ikke å følge algoritmene bestemt og nøyaktig, noe som gjør at den PRO.AT kommer på et høyere nivå under algoritmebehandling. Den INS.AT og den PRO.AT har fra min egen analyse ingen likhetstrekk og deler ingen av de samme kategoriene.

Dermed viser sammenhengen mellom mine funn at den prosedurale algoritmiske tenkeren og den konseptuelle algoritmiske tenkeren til sammen viser flest tegn på algoritmisk tenkning. En antagelse er at disse elevene til *sammen* hadde klart å løse oppgaven. Den PRO.AT hadde klart å følge algoritmene korrekt, mens den KON.AT kunne bidratt med det faglige aspektet ved oppgaven.

Sammenhengen mellom den PRO.AT og den KON.AT er illustrert i tabellen under.

Tabell 3:Tabellen viser at den prosedurale algoritmiske tenkeren (PRO.AT) og den konseptuelle algoritmiske tenkeren (KON.AT til sammen viser flest kjennetegn på algoritmisk tenkning.

	Abstraksjon	Generalisering	Algoritmebehandling
Nivå 0	INS.AT	INS.AT PRO.AT	INS.AT
Nivå 1	PRO.AT KON.AT	KON.AT	KON.AT
Nivå 2			PRO.AT

Kapittel 6

6 Avslutning

Problemstillingen for denne masteroppgaven er: *Hva kjennetegner elevers algoritmiske tenkning når de jobber med geometri i blokkprogrammering?* For å svare på denne problemstillingen har jeg undersøkt hvordan kjennetegnene *abstraksjon, generalisering* og *algoritmebehandling* kommer til uttrykk når 10 elever på 6.trinn arbeider med geometri i blokkbasert programmering. Oppgaven som elevene får utdelt er uten bruk av digitale verktøy.

6.1 Hva har jeg funnet ut?

Studiens funn viser tre typer algoritmiske tenkere: «*Den instrumentelle algoritmiske tenkeren*», «*Den prosedurale algoritmiske tenkeren*» og «*Den konseptuelle algoritmiske tenkeren*». Bakgrunnen for disse er hvordan kjennetegnene abstraksjon, generalisering og algoritmebehandling kommer til uttrykk i elevenes løsningsforslag.

I diskusjonskapittelet konkluderer jeg med at den prosedurale algoritmiske tenkeren og den konseptuelle algoritmiske tenkeren til sammen viser flest kjennetegn på algoritmisk tenkning. Grunnen til dette er at disse elevene behersker et mer abstrakt programmeringsspråk (Bocconi et al., 2016). Abstraksjon igjen er det viktigste kjennetegnet på algoritmisk tenkning. Abstraksjon vil være viktig for å lykkes med programmering fordi alle programmeringsspråk er abstraksjoner av den virkelige verden (Wing, 2008). Abstraksjon er igjen en viktig forløper til generalisering fordi en viktig del av å programmere er å utvikle generelle algoritmer (Bocconi et. al., 2016). Den konseptuelle algoritmiske tenkeren viser tegn til generalisering gjennom å oppdage likheter og ulikheter i funksjonsblokkene. Dette gjør det også lettere å løse liknende problem. Den prosedurale algoritmiske tenkeren viser det høyeste nivå på algoritmebehandling fordi disse elevene klarer å følge algoritmene på en bestemt og tydelig måte (Bocconi et. al., 2016; Hiebert og Lefevere, 1986).

Dermed kjennetegnes den algoritmiske tenkeren av å både ha en prosedural og konseptuell forståelse i matematikk, men også i programmering. Den prosedurale algoritmiske tenkeren klarer å følge algoritmene nøyaktig og bestemt. Dette vil være en viktig ferdighet for å klare å forstå *hvordan* en algoritme fungerer. Den konseptuelle algoritmiske tenkeren klarer i større

grad å reflektere over det matematiske problemet som skal løses og bruker mindre tid på å følge algoritmene nøyaktig. Dette vil være en viktig ferdighet for å klare å forstå *hvorfor* en algoritme fungerer.

6.2 Teoretiske og didaktiske implikasjoner

Hvilke teoretiske implikasjoner har min studie? Det er utfordrende å plassere studien min inn i tidligere forskning fordi algoritmisk tenkning fortsatt er et nytt begrep i undervisningssammenheng. Det meste jeg har lest om algoritmisk tenkning er teoretiske betraktninger og definisjoner av begrepet, noe som kan skyldes at algoritmisk tenkning ofte blir nevnt i sammenheng med programmering og en rekke andre begrep som er viktig for programmeringsfaget. Av norske forskningsartikler er det bare Gjøvik og Torkildsen (2019) som har gått grundigere inn på å prøve å forstå begrepet algoritmisk tenkning, dette har vært et viktig utgangspunkt for å kunne skrive denne oppgaven. Også Knuth (1985) skriver om algoritmisk tenkning og matematisk tenkning, men ikke i sammenheng med undervisning. Det eneste jeg kan støtte mine funn på er at Seymour Papert brukte algoritmisk tenkning synonymt med prosedural kunnskap når han innførte Logo på 1960-tallet (Grover & Pea, 2013). Jeg har ikke lyktes med å finne noen artikler eller annen forskningslitteratur innenfor programmering som ser på algoritmisk tenkning i lys av konseptuell forståelse.

Hvilke didaktiske implikasjoner har min studie? Dersom algoritmisk tenkning kjennetegnes av at elever både må utvikle en prosedural og konseptuell forståelse i både matematikk og programmering, kan det tyde på at programmering vil bli krevende for enkelte elever. Hvordan vil elever takle et helt nytt notasjonssystem? Hvordan vil det gå med elever som fra før av sliter med abstraksjon i matematikk? Kan programmering forsterke en instrumentell forståelse i matematikk? Kan programmering forsterke en konseptuell eller prosedural forståelse i matematikk? Gjøvik og Torkildsen (2019) skriver at algoritmisk tenkning må være noe mer enn å bare anvende standardalgoritmer i matematikk. Det må handle om å lære matematikk men med hjelp av programmering. Jeg har med denne oppgaven vist at det er mulig å utforske algoritmisk tenkning gjennom analog programmering, men det er viktig å understreke at jeg ikke mener at matematikklærere skal frata seg ansvaret for å gi elever opplæring i ulike programmeringsverktøy og programmeringsspråk. En måte å jobbe med algoritmisk tenkning på kan være å ta i bruk Bocconi et al. (2016) sitt rammeverk. Dette rammeverket konkretiserer et nokså abstrakt begrep.

6.3 Hvordan forske videre på dette?

En masteroppgave skal ifølge Postholm et al. (2018, s. 245) bidra til forskningsintensitet og kunnskapsutviklet i skolen. For å forske videre på dette ville jeg undersøkt i en større skala hvordan elever takler overgangen fra matematikkfaget til programmeringsfaget. Ikke minst hadde det vært interessant å undersøke om elever med en instrumentell, prosedural eller konseptuell forståelse i matematikk også tilnærmer seg programmeringsfaget på samme måte. Dermed har jeg utformet en ny problemstilling som det kunne vært interessant å forske videre på:

Hvordan kan programmering påvirke elevers matematiske forståelse?

Referanseliste

Adler, P. A. & Adler, P. (1994). Observational techniques.

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., Engelhardt, K., Kampylis, P. & Punie, Y. (2016). Developing Computational Thinking in Compulsory Education. Implications for policy and practice. *EUR - Scientific and Technical Research Reports*. 10.2791/792158

Bocconi, S., Chioccariello, A. & Earp, J. (2018). *THE NORDIC APPROACH TO INTRODUCING COMPUTATIONAL THINKING AND PROGRAMMING IN COMPULSORY EDUCATION*.

Braun, V. & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative research in psychology, 3*(2), 77-101.

Brousseau, G. (2006). *Theory of didactical situations in mathematics: Didactique des mathématiques, 1970–1990* (bd. 19): Springer Science & Business Media.

Bueie, H. (2019). *Programmering for matematikklærere*. Oslo: Universitetsforlaget.

Burton, D. M. (1985). The history of mathematics: An introduction. *Group, 3*(3), 35.

Christoffersen, L. & Johannessen, A. (2012). *Forskningsmetode for lærerutdanningene*: Abstrakt.

Creswell, J. W. (2013). *Qualitative inquiry and research design: Choosing among five traditions*: Sage Publications, Inc.

Creswell, J. W. (2014). *Research design*: Sage publications Thousand Oaks, CA.

Cuoco, A., Goldenberg, E. P. & Mark, J. (1996). Habits of mind: An organizing principle for mathematics curricula. *The Journal of Mathematical Behavior, 15*(4), 375-402.

Davis, P., Hersh, R. & Marchisotto, E. A. (2011). *The mathematical experience*: Springer Science & Business Media.

Dicks, B., Mason, B., Coffey, A. & Atkinson, P. (2006). *Qualitative research and hypermedia: Ethnography for the digital age*: Sage.

Drijvers, P., Kieran, C., Mariotti, M.-A., Ainley, J., Andresen, M., Chan, Y. C., . . .Leung, A. (2009). Integrating technology into mathematics education: Theoretical perspectives. I *Mathematics education and technology-rethinking the terrain* (s. 89-132): Springer.

Elo, S. & Kyngäs, H. (2008). The qualitative content analysis process. *Journal of advanced nursing, 62*(1), 107-115.

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

Gold, R. L. (1957). Roles in sociological field observations. *Soc. F., 36*, 217.

Grover, S. & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational researcher, 42*(1), 38-43.

Grønmo, S. & Hovde, K.-O. (2020). Algoritme. Hentet 8. mai 2021 fra Hentet fra https://snl.no/algoritme

Haraldsrud, A. D., Sveinsson, H. A. & Løvold, H. H. (2020). *Programmering i skolen*. Oslo: Universitetsforlaget.

Hiebert, J. & Lefevre, P. (1986). Conceptual and procedural knowledge in mathematics: An introductory analysis. I *Conceptual and procedural knowledge: The case of mathematics.* (s. 1-27). Hillsdale, NJ, US: Lawrence Erlbaum Associates, Inc.

Istad, R. M. & Kristoffersen, B. (2019). *Forstå programmering : med Java* (2. utg. utg.). Oslo: Universitetsforl.

Kaufmann, O. T. & Stenseth, B. (2020). Programming in mathematics education. *International Journal of Mathematical Education in Science and Technology*, 1-20. 10.1080/0020739x.2020.1736349

Knuth, D. E. (1985). Algorithmic thinking and mathematical thinking. *The American Mathematical Monthly, 92*(3), 170-181.

Lytle, N., Cateté, V., Boulden, D., Dong, Y., Houchins, J., Milliken, A., . . .Barnes, T. (2019). *Use, Modify, Create: Comparing Computational Thinking Lesson Progressions for STEM Classes.* Foredrag holdt ved Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education.

Meld. St.20 (2018). *Fornyer innholdet i skolen*. Hentet fra https://www.regjeringen.no/no/aktuelt/fornyer-innholdet-i-skolen/id2606028/

NAOB.). Algoritmisk. (bd. 2021). Oslo: Det Norske Akademi for Språk og Litteratur. Hentet 19.mars 2021 fra Hentet fra https://naob.no/ordbok/algoritmisk

NESH. (2016). Forskningsetiske retningslinjer for samfunnsvitenskap, humaniora, juss og teknologi. Hentet fra https://www.etikkom.no/forskningsetiske-retningslinjer/Samfunnsvitenskap-jus-og-humaniora/Forord/

NOU 2013:2. *Hindre for digital verdiskaping*

Hentet fra https://www.regjeringen.no/no/dokumenter/nou-2013-2/id711002/

Postholm, M. B., Jacobsen, D. I. & Søbstad, R. (2018). *Forskningsmetode for masterstudenter i lærerutdanningen* (Forskningsmetode). Oslo: Cappelen Damm akademisk.

Ryan, A. B. (2006). Post-positivist approaches to research. *Researching and Writing your Thesis: a guide for postgraduate students*, 12-26.

Savin-Baden, M. & Howell-Major, C. (2013). Qualititative research: The essential guide to theory and practice. *Qualitative Research: The Essential Guide to Theory and Practice. Routledge*.

Seale, C. (1999). *Quality in qualitative research* (Qualitative inquiry, bd. 5).

Silverman, D. (2011). *Interpreting qualitative data. A Guide to the Principles of Qualitative Research*: Sage Publications, London.

Skemp, R. R. (1976). Relational understanding and instrumental understanding. *Mathematics teaching, 77*(1), 20-26.

Stenseth, B., Kaufmann, O. T. & Forsstöm, S. E. (2019). Programmering og matematikk. *Tangenten, 30*(2), 7-12. http://www.caspar.no/tangenten/2019/tangenten%202%202019%20Stenseth%20et%20al.pdf

Svartdal, F. (2009). *Psykologiens forskningsmetoder* (3. utg. utg.). Bergen: Fagbokforl.

Svartdal, F. (2018, 23. mars). Tenkning. I *Store norske leksikon*.Hentet fra https://snl.no/tenkning

Thagaard, T. (2013). *Systematikk og innlevelse : en innføring i kvalitative metoder* (5. utg. utg.). Bergen: Fagbokforl.

Utdanningsdirektoratet. (2020a). *Algoritmisk tenkning*. Hentet fra https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/

Utdanningsdirektoratet. (2020b). *Kvalitet og kompetanse - profesjonsfalig digital kompetanse*. Hentet fra https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/

Utdanningsdirektoratet. (2020c). *Læreplan i matematikk* (MAT01-05).

Van de Walle, J. A., Bay-Williams, J. M., Lovin, L. H. & Karp, K. S. (2013). Shapes and Properties. I *Teaching student-centered mathematics : developmentally appropriate instruction for grades 6-8*

(2nd ed. utg.): Pearson.

vitensenter. (2018). Begrepsforklaringer. Hentet 10.mars fra https://www.vitensenter.no/superbit/begrepsforklaringer/

Wing. (2006). Computational thinking. *Communications of the Acm, 49*(3), 33-35. Doi 10.1145/1118178.1118215

Wing. (2008). Computational thinking and thinking about computing. *366*, 1-1. 10.1109/IPDPS.2008.4536091

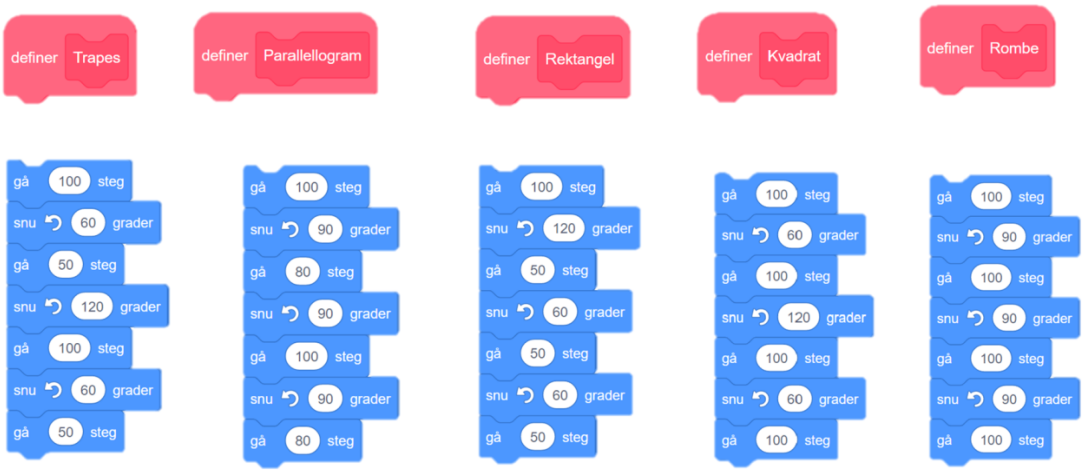
Figur og tabell

Tabell 1 Den egyptiske multiplikasjonsalgoritmen, hentet fra Bueie, H. (2019). Programmering for matematikklærere. Oslo: Universitetsforlaget.

Figur 20 Bocconi et al. (2016) sine seks kjennetegn på computational thinking (t.v). Gjøvik og Torkildsen sine norske oversettelser (t.h), hentet Gjøvik, Ø. & Torkildsen, H. A. (2019). Algoritmisk tenkning. *Tangenten–tidsskrift for matematikkundervisning*, *30*(3), 31-37.

Vedlegg 1: Oppgaven

Scratch har vært uheldig å glemt å lagre programmet sitt. Kan dere hjelpe Scratch å sortere algoritmene slik at de havner under de rette firkantene?



Vedlegg 2: Informasjonsskriv og samtykkeskjema

Vil du delta i forskningsprosjektet

«Argumentasjon og programmering – en kvalitativ studie om hva som kjennetegner elevers argumentasjoner når de arbeider med geometriske figurer i programmeringsspråket Scratch»

Dette er et spørsmål til deg om å delta i et forskningsprosjekt hvor formålet er å samle inn data til min masteroppgave i matematikkdidaktikk. Masteroppgaven handler om hva som kjennetegner elevers argumentasjoner når de arbeider med todimensjonale figurer i programmeringsspråket Scratch. I dette skrivet gir jeg deg informasjon om målene for prosjektet og hva deltakelse vil innebære for deg.

Formål
I min masteroppgave ønsker jeg å analysere hvordan elever argumenterer når de arbeider i programmeringsspråket Scratch. Scratch er et blokkbasert programmeringsspråk som egner seg godt til å lage geometriske figurer. Det er intuitivt, visuelt, og mye brukt i skolen i dag.

Aktuell problemstilling:
Hva kjennetegner elevers argumentasjoner når de arbeider med todimensjonale figurer i programmeringsspråket Scratch?

Metodene for datainnsamling er følgende:

- Videoobservasjon: dette er for å se hvordan elevene arbeider i Scratch.
- Lydopptak: dette er for å høre hvordan elevene diskuterer underveis.
- Skjermopptak uten lyd: kun aktuelt dersom det blir vanskelig å filme skjermene med GoPro-kamera.

Omfanget vil være på 4x45 min

- 2x45 min: Introduksjonskurs i Scratch.
- 2x45 min: Løse oppgaver i Scratch.

Hvem er ansvarlig for forskningsprosjektet?

- UIT Norges Arktiske Universitet – Fakultet for humanoria, samfunnsvitenskap og lærerutdanning.
- Jan Nyquist Roksvold (veileder for masterprosjektet og førsteamanuensis i matematikkdidaktikk ved institutt for lærerutdanning og pedagogikk).

Hvorfor får du spørsmål om å delta?
Du får spørsmål om å delta i mitt masterprosjekt fordi du går i 6.klasse og fordi masteroppgaven min tar utgangspunkt i et kompetansemål etter 6.trinn: «Bruke variabler, løkker, vilkår og funksjoner i programmering til å utforske geometriske figurer og mønstre».

I dette masterprosjektet har jeg 5 GoPro-kameraer til disposisjon, dermed er dere 10 elever som får spørsmål om å delta i studiet mitt. Dere skal arbeide sammen to og to.

Hva innebærer det for deg å delta?
For å delta i dette studiet trenger jeg ditt samtykke for å få lov til å gjennomføre videoobservasjon og lydopptak av deg. Du kommer bestandig til å sitte sammen med en annen medelev og diskutere. Du må også godkjenne å bruke 4x45 min av din normale undervisning til å delta i mitt masterprosjekt. Dersom foreldrene dine ønsker mer informasjon er det bare å ta kontakt.

Det er frivillig å delta
Det er frivillig å delta i prosjektet. Hvis du velger å 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 hvis du ikke vil delta eller senere velger å trekke deg.
Dersom du ønsker å trekke deg er det bare å ta kontakt med meg eller læreren din.

Ditt personvern – hvordan vi oppbevarer og bruker dine opplysninger
Jeg vil bare bruke opplysningene om deg til formålene jeg har fortalt om i dette skrivet. Jeg behandler opplysningene konfidensielt og i samsvar med personvernregelverket.
Dette betyr blant annet at navn og kontaktopplysninger vil erstattes med fiktive navn. Alt av datamateriale vil bli lagret innelåst og passordbeskyttet.
Andre som vil ha tilgang til dine opplysninger er:

1. Jan Roksvold (veileder for masterprosjektet).

Merk at det kun er oss to som vil ha tilgang til dine opplysninger. Det vil ikke være mulig å spore informasjonen i masteroppgaven tilbake til deg som deltaker.

Hva skjer med opplysningene dine når vi avslutter forskningsprosjektet?
Opplysningene anonymiseres når prosjektet avsluttes/oppgaven er godkjent, noe som etter planen er 15.mai 2020. Datamaterialet og dine personopplysningen vil da bli fjernet.

Dine rettigheter
Så lenge du kan identifiseres i datamaterialet, har du rett til:
- innsyn i hvilke personopplysninger som er registrert om deg, og å få utlevert en kopi av opplysningene,
- å få rettet personopplysninger om deg,
- å få slettet personopplysninger om deg, og
- å sende klage til Datatilsynet om behandlingen av dine personopplysninger.

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

På oppdrag fra UIT 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:
- Jan Roksvold , e-post: jan.n.roksvold@uit.no, tlf: 77646141.
- Elisabeth Dahl Olafsen, e-post: eol057@uit.no, tlf: 97739722

- Vårt personvernombud: Joakim Bakkevold, e-post: personvernombud@uit.no, tlf: 776 46 322 og 976 915 78

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

Prosjektansvarlig	**Student**
Jan Roksvold	Elisabeth Dahl Olafsen

Samtykkeerklæring

Jeg har mottatt og forstått informasjon om prosjektet *«begrepsforståelse i geometri»*, og har fått anledning til å stille spørsmål. Jeg samtykker til:

- ☐ å delta å bli tatt video av.
- ☐ å delta i å bli tatt lydopptak av.
- ☐ å levere inn håndskrevne notater.

Jeg samtykker til at mine opplysninger behandles frem til prosjektet er avsluttet

---

(Signert av prosjektdeltaker, dato)

Vedlegg 3: Godkjennelse fra NSD

NSD NORSK SENTER FOR FORSKNINGSDATA

NSD sin vurdering

Prosjekttittel

Argumentasjon og programmering - en kvalitativ studie om elevers argumentasjoner når de arbeider geometrioppgaver i programmeringsspråket Scratch.

Referansenummer

418616

Registrert

05.01.2021 av Elisabeth Dahl Olafsen - eol057@post.uit.no

Behandlingsansvarlig institusjon

UiT Norges Arktiske Universitet / Fakultet for humaniora, samfunnsvitenskap og lærerutdanning / Institutt for lærerutdanning og pedagogikk

Prosjektansvarlig (vitenskapelig ansatt/veileder eller stipendiat)

Jan Roksvold, jan.n.roksvold@uit.no, tlf: 77646141

Type prosjekt

Studentprosjekt, masterstudium

Kontaktinformasjon, student

Elisabeth Dahl Olafsen, eol057@uit.no, tlf: 97739722

Prosjektperiode

10.01.2021 - 15.05.2021

Status

25.01.2021 - Vurdert

Vurdering (1)

25.01.2021 - Vurdert

Det er vår vurdering at behandlingen av personopplysninger i prosjektet vil være i samsvar med personvernlovgivningen så fremt den gjennomføres i tråd med det som er dokumentert i meldeskjemaet 25.01.2021 med vedlegg, samt i meldingsdialogen mellom innmelder og NSD. Behandlingen kan starte.

MELD VESENTLIGE ENDRINGER
Dersom det skjer vesentlige endringer i behandlingen av personopplysninger, kan det være nødvendig å melde dette til NSD ved å oppdatere meldeskjemaet. Før du melder inn en endring, oppfordrer vi deg til å lese om hvilke type endringer det er nødvendig å melde:

https://nsd.no/personvernombud/meld_prosjekt/meld_endringer.html

Du må vente på svar fra NSD før endringen gjennomføres.

TYPE OPPLYSNINGER OG VARIGHET
Prosjektet vil behandle alminnelige kategorier av personopplysninger frem til 15.05.2021.

LOVLIG GRUNNLAG
Prosjektet vil innhente samtykke fra foresatte til behandlingen av personopplysninger om barna/elevene. Vår vurdering er at prosjektet legger opp til et samtykke i samsvar med kravene i art. 4 og 7, ved at det er en frivillig, spesifikk, informert og utvetydig bekreftelse som kan dokumenteres, og som foresatte kan trekke tilbake. Barna/elevene vil også samtykke til deltakelse.

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

PERSONVERNPRINSIPPER
NSD vurderer at den planlagte behandlingen av personopplysninger vil følge prinsippene i personvernforordningen om:

- lovlighet, rettferdighet og åpenhet (art. 5.1 a), ved at de registrerte får tilfredsstillende informasjon om og samtykker til behandlingen
- formålsbegrensning (art. 5.1 b), ved at personopplysninger samles inn for spesifikke, uttrykkelig angitte og berettigede formål, og ikke behandles videre til nye uforenlige formål
- dataminimering (art. 5.1 c), ved at det kun behandles opplysninger som er adekvate, relevante og nødvendige for formålet med prosjektet
- lagringsbegrensning (art. 5.1 e), ved at personopplysningene ikke lagres lengre enn nødvendig for å oppfylle formålet

DE REGISTRERTES RETTIGHETER
NSD vurderer at informasjonen om behandlingen som de registrerte og deres foresatte vil motta oppfyller lovens krav til form og innhold, jf. art. 12.1 og art. 13.

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

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

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

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

OPPFØLGING AV PROSJEKTET
NSD vil følge opp ved planlagt avslutning for å avklare om behandlingen av personopplysningene er avsluttet/pågår i tråd med den behandlingen som er dokumentert.

Lykke til med prosjektet!

Kontaktperson hos NSD: Marie Strand Schildmann
Tlf. Personverntjenester: 55 58 21 17 (tast 1)