Programmering som en matematisk representasjon

En kvalitativ studie av R2-elevers arbeid med programmering

Sondre Waage Kolbeinsen

Masteroppgave i matematikkdidaktikk MAT399K

Matematisk institutt

UNIVERSITETET I BERGEN

01. juni 2023

Forord

Tiden flyr når man har det gøy, sies det. Kanskje det er derfor disse fem årene på lektorutdanningen føles å ha gått så utrolig fort? Høsten 2018 satte jeg min første fot inn på Realfagbygget på Universitetet i Bergen. Våren fem år senere, skriver jeg dette forordet på min masteroppgave som markerer slutten på en helt fantastisk studietid.

Det å skrive en masteroppgave har vært en akademisk reise som jeg ikke hadde klart å gjennomføre hvis det ikke hadde vært for de gode menneskene jeg har vært omringet av. På denne reisen har det vært flere involverte som jeg ønsker å uttrykke min takknemlighet til. Jeg vil starte med å takke læreren jeg har samarbeidet med i gjennomføringen av datainnsamlingen. Du tok meg imot med åpne armer, viste interesse for prosjektet, og lot meg låne din klasse for å samle inn data. Her må det også rettes en stor takk til elevene som samtykket til å bidra i min studie.

Jeg vil også rette en stor takk til min dyktige veileder Jan Nyquist Roksvold, som har vært støttende under hele prosessen. Dine tilbakemeldinger og innspill har vært til god hjelp i en ellers krevende skriveperiode. Jeg vil også takke Johan Lie og Stian Hirth i 4. etasje. Takk for at dere alltid har tatt dere tid til en prat når jeg har hatt behov for å lufte tanker rundt oppgaven. Jeg må også takke alle dere andre didaktikere som og har hjulpet med å svare på spørsmål til prosjektet mitt.

En helt essensiell ingrediens når en skal skrive masteroppgave er gode medstudenter. Jeg vil takke for at vi har hatt et så godt miljø i løpet av studietiden. Takk for at dere har bidratt til morsomme, lange og sosiale pauser. En spesiell takk må rettes til alle mine lektorboys. Lektorstudiet hadde ikke vært det samme uten dere! Jeg vil også rette en takk til «kokkene» i 4. etasje som har foret oss med vafler og kaffe hver torsdag.

Til slutt vil jeg takke familie, kjæreste og venner. Takk til mamma, pappa og lillebror Vegard for deres støtte gjennom hele studieløpet mitt. Takk til Julie som har stått klar med middag når jeg har kommet hjem etter lange dager på lesesalen. Dere er best!

Bergen, juni 2023

Sondre Waage Kolbeinsen

Sammendrag

I Kunnskapsløftet 2020 er matematiske representasjoner og programmering to viktige temaer. Representasjoner, sammen med kommunikasjon, danner et eget kjerneelement til matematikkfaget. Sammen med innføringen av Kunnskapsløftet 2020 fikk programmering en egen plass i de matematiske kompetansemålene. Programmering kan derfor anses som en relativt fersk nykommer i norske matematikklasserom. Disse to temaene inngår i denne masteroppgaven som undersøker problemstillingen: *Programmering som en matematisk representasjon*. Problemstillingen blir belyst ved hjelp av forskningsspørsmålene (1) *Hvordan kan programkode betraktes som en matematisk representasjon?* og (2) *Hvordan kan programmering støtte opp under elevers matematiske verbale representasjon?*

Til prosjektet har det blitt brukt et kvalitativt forskningsdesign med lyd-, video- og skjermopptak som metode. Forskningen har blitt gjennomført i én matematikklasse med R2-elever som deltakere. Opptak av elevgrupper som arbeid med en matematisk programmeringsoppgave om numerisk integrasjon, danner datamaterialet for studien. Elevene har brukt det tekstbaserte programmeringsspråket Python i gjennomførelsen av oppgaven. Datamaterialet har blitt analysert ved transkribering av lydopptak, kommentarfunksjoner i Microsoft Word og strukturering av elevers verbale representasjoner i tabeller basert på matematiske objekter.

I oppgaven presenteres det fire funn som har blitt gjort: (1) elever behandler programkode som noe manipulativt, (2) elever forstår ikke nødvendigvis symbolbruk i Python, (3) elever skriver programkode for seg selv og (4) programmeringsobjekter og programmeringsbegreper kan mediere elevenes verbale representasjoner av matematiske objekter. Disse blir drøftet opp mot problemstillingen.

Det ser ut til å være lite tidligere forskning på programmering som en matematisk representasjon. Dette kan indikere at det er et eksisterende gap i forskningsfeltet, som min studie kan bidra å fylle. I oppgaven presenteres også en teoretisk implikasjon knyttet til en tanke om at programmering kan anses som en egen representasjonsform, sammen med allerede etablerte matematiske representasjonsformer.

Innholdsfortegnelse

1.0 Innledning

Innledningsvis vil jeg starte med å presentere bakgrunn og behov for valg av tema før jeg videre presenterer oppgavens problemstilling og forskningsspørsmål. I slutten av innledningen legger jeg frem hvordan resten av masteroppgaven er strukturert.

1.1 Bakgrunn og behov

Når vi arbeider med matematikk, er vi omgitt av ulike matematiske representasjoner. Matematikk er abstrakt av natur, men med hjelp av representasjoner får vi en slags tilgang til matematiske objekter. Dette medfører også at matematiske representasjoner får en sentral rolle i matematikklasserommet. I det matematikkdidaktiske tidsskriftet Tangenten blir det fortalt at det å kunne forstå og bruke ulike representasjoner er en viktig del av matematisk kompetanse (Enge & Valenta, 2013, s. 8). Med Kunnskapsløftet 2020 (videre forkortet som LK20) har representasjoner, sammen med kommunikasjon, blitt et eget kjerneelement i matematikkfaget.

Med LK20 har også programmering blitt en integrert del av skolehverdagen. Det å kunne «bruke programmering» blir nevnt eksplisitt i enkelte kompetansemål i matematikkfaget. Selv om det i dag er tre år siden LK20 ble innført, har jeg selv opplevd at det er flere elever og lærere som fremdeles har lite erfaring med, eller unngår, programmering i matematikkfaget. Matematiske representasjoner har derimot alltid vært en viktig del av matematikkfaget. I denne masteroppgaven kombinerer jeg disse to temaene; programmering og representasjoner, og undersøker programmering som en matematisk representasjon.

Mye av motivasjonen for valget av dette temaet springer ut ifra dets relevans i matematikkdidaktikken. I og med at programmering er en relativt fersk nykommer i skolematematikken, mener jeg det er viktig å undersøke hvordan det henger sammen med andre elementer i læreplanen, i dette tilfellet matematiske representasjoner.

Valget av dette temaet er også basert på personlig motivasjon. Mitt første møte med det tekstbasert programmeringsspråket Python, var når jeg tok innføring i programmering som et valgfag på Universitetet i Bergen. Helt fra starten av semesteret var jeg engasjert, og det å gjøre obligatoriske innleveringer har aldri vært så gøy. Året etter tok jeg et fag innenfor matematikkdidaktikk som omhandlet representasjoner og problemløsning i matematikkundervisningen. Her gikk det opp for meg hvor stor nytteverdi matematiske representasjoner har i læring av matematikk.

I planleggingen av problemstillingen for masteroppgaven visste jeg derfor at jeg ville se nærmere på disse to temaene: programmering og matematiske representasjoner.

1.2 Tidligere forskning

I arbeid med dette aktuelle temaet har jeg forsøkt å finne tidligere forskning for å sette mitt eget prosjekt i en bredere sammenheng. Det viser seg at mengden av litteratur som undersøker programmering som en matematisk representasjon er begrenset. Dette kan indikere at det er et eksisterende gap i forskningsfeltet, som min studie kan bidra å fylle. Samtidig kan dette gapet påvirke studien min i den forstand at det begrenser muligheten for å sammenligne og kontrastere mine funn opp mot tidligere forskning.

1.3 Problemstilling og forskningsspørsmål

Basert på bakgrunn og behov for temaene som har blitt nevnt, har jeg en bearbeidet en problemstilling, og to forskningsspørsmål som skal bidra å belyse problemstillingen. I denne studien undersøker jeg problemstillingen:

Programmering som en matematisk representasjon.

Programmering er et samlebegrep for flere elementer som inngår i det *å programmere* (dette blir også belyst i delkapittel 2.2.1). Når man programmerer skriver man gjerne en programkode. I et tekstbasert programmeringsspråk vil dette være linjer sammensatt av symboler som kan tolkes av en datamaskin. Det *å skrive programkode* er altså noe som inngår i programmering. Når elever er ferdig med en matematisk programmeringsoppgave, har de vanligvis produsert en programkode. Jeg ville undersøke hvordan denne programkoden kunne bli betraktet som en matematisk representasjon, noe som resulterte i forskningsspørsmål 1:

Forskningsspørsmål 1: Hvordan kan programkode betraktes som en matematisk representasjon?

En av de vanligste representasjonsformene vi omgir oss med, er den verbale representasjonsformen. Når en lærer underviser, eller når vi snakker med hverandre, benytter vi oss av verbale representasjoner. Verbale representasjoner er også eksterne, noe som gjør det mulig for andre å observere. Forskningsspørsmål 2 undersøker elevenes verbale representasjoner i arbeid med programmering:

Forskningsspørsmål 2: Hvordan kan programmering støtte opp under elevers matematiske verbale representasjon?

Forskningsspørsmålene vil bidra til å gi ulike perspektiver av programmering som en matematisk representasjon.

1.4 Oppgavens struktur

Oppgaven er delt inn i seks hovedkapitler med tilhørende delkapitler. Hovedkapitlene er strukturert på følgende måte: (1) innledning, (2) teori, (3) metode og analyse, (4) resultat, (5) diskusjon og (6) avslutning.

I kapittel 2 presenterer jeg relevant teori for oppgaven. Her legger jeg frem relevant teori til matematiske representasjoner og programmering. Jeg ser også på to artikler med tidligere forsking som inkluderer begge temaene programmering og representasjoner.

I kapittel 3 presenterer jeg prosjektets forskningsdesign og metode. Her forteller jeg blant annet om informantvalg og gjennomføring av datainnsamling. Deretter presenterer jeg analyseprosessen av dataene, før jeg til slutt ser på studiens kvalitet og etiske perspektiver.

I kapittel 4 legger jeg frem fire funn som har blitt gjort på bakgrunn av analyseprosessen. Funnene underbygges med utdrag fra datamaterialet.

I kapittel 5 diskuterer jeg funnene fra kapittel 4 ved å se på mulige årsaker og implikasjoner. Her vil jeg trekke inn relevant teori fra kapittel 2. Funnen vil også bli diskutert opp mot problemstillingen.

Kapittel 6 er det avsluttende kapittelet. Her vil jeg gi en oppsummerende konklusjon til forskningsspørsmålene og problemstillingen jeg har undersøkt i oppgaven. Jeg vil også fortelle om studiens teoretiske og praktiske implikasjoner, før jeg til slutt skriver om veien videre og kommer med forslag til videre forskning på feltet.

2.0 Teori

I dette kapittelet vil jeg presentere relevant teoretisk bakgrunn og tidligere forskning som er relevant for å belyse problemstillingen. Kapittelet er delt inn i tre deler. I første del legger frem teori om matematiske representasjoner. I andre del legger jeg fram teori tilknyttet programmering. I tredje og siste del vil jeg presentere tre artikler som inneholder både temaene programmering og representasjoner, hvor to av disse artiklene er tidligere forskning.

2.1 Matematiske representasjoner

Goldin (2020) skriver i «Encyclopedia of Mathematics Education» om matematiske representasjoner. Her starter han med å legge frem en forklaring av hva matematisk representasjoner kan være:

> As most commonly interpreted in education, mathematical representations are visible or tangible productions – such as diagrams, number lines, graphs, arrangements of concrete objects or manipulatives, physical models, written words, mathematical expressions, formulas and equations, or depictions on the screen of a computer or calculator – that encode, stand for, or embody mathematical ideas or relationships. (Goldin, 2020, s. 566)

Det nevnes i starten av sitatet at denne forklaringen er basert på hvordan matematiske representasjoner *vanligvis* blir tolket i et undervisningsperspektiv. Matematiske representasjoner er i denne forstand synlige eller håndgripelige produkter som omformer, står for, eller legemliggjør matematiske ideer og sammenhenger (Goldin, 2020, s. 566).

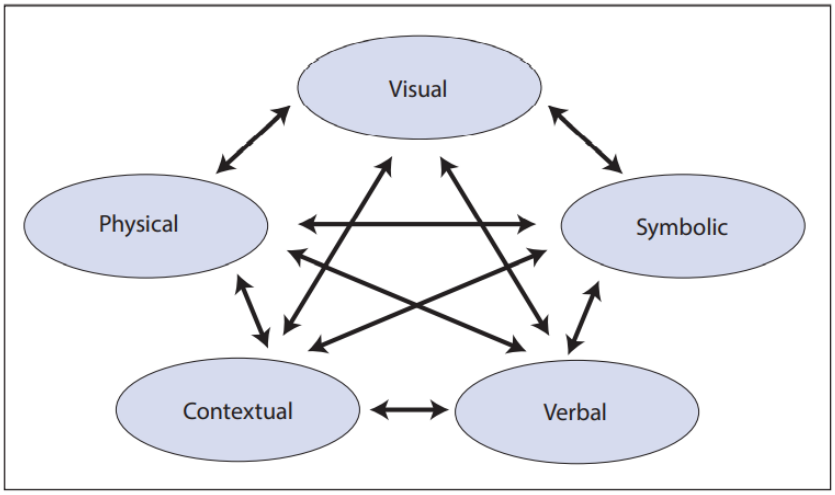
Matematiske representasjoner kan ifølge Goldin (2020) være *interne* eller *eksterne*. Når vi snakker om representasjoner som er interne for et individ, referer representasjonen til en persons mentale eller kognitive konstruksjoner, konsepter eller konfigurasjoner (Goldin, 2020, s. 567). Interne representasjoner omtaler jeg derfor på lik linje som mentale, kognitive eller indre representasjoner. Disse representasjonene innebærer for eksempel et individs visuelle og/eller spatiale kognitive representasjon av geometriske objekter eller matematiske mønstre, operasjoner eller situasjoner; deres kinestetiske koding av operasjoner, former og bevegelser; deres interne konseptuelle modeller av matematiske ideer; språket de bruker internt for å beskrive matematiske situasjoner; og deres heuristiske planer og strategier for problemløsning (Goldin, 2020, s. 567). Interne representasjoner har den egenskapen at de kun er tilgjengelig for individet selv, og kan ikke observeres av andre.

I motsetning til interne representasjoner finnes det også eksterne representasjoner. Disse representasjonene er ikke tilgjengelig kun for individet selv, men også for andre til å observere

Goldin (2020, s. 566). Dette kan for eksempel være vårt naturlige språk, skrevne symboler eller en tegning. Matematiske representasjoner kan derfor være noe man produserer eksternt, eller konstruerer internt, og sammenhengen mellom disse to. Med andre ord kan representasjon henvise til noe man *gjør* (Goldin, 2020, s. 567).

Goldin (2020, s. 568) forteller også om hvordan representasjoner som oppstår i undervisningskontekst ofte er uferdige og nesten alltid tvetydige. Dette skiller seg ut fra det han kaller for «conventinal structured mathematical representatsional systems». Dette kan oversettes til norsk som «tradisjonelle matematiske representasjoner», som betyr at de baserer seg på oppfatninger som deles av et større matematisk fellesskap og som kan uttrykkes mer i detalj. Et eksempel på tradisjonell matematisk representasjon kan være tegnene [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] innenfor nummerering (Goldin, 2020, s. 568). Når elever derimot lager sine egne matematiske representasjoner, vil disse ofte være det Goldin (2020, s. 567) kaller for *idiosynkratiske*, altså at det representeres noe særegent som gjerne gjelder kun den enkelte elev.

Lesh et al. (1987, s. 1) identifiserer fem distinkte representasjonsformer som oppstår i matematisk læring og problemløsning. «National Council of Mathematics» (NCTM, 2014, s. 24) omtaler disse fem som «the general classification scheme for types of representations». Det er disse fem representasjonsformene som vanligvis blir presentert når man snakker om representasjoner. Lesh et al. (1987, s. 2) sier også at disse representasjonsformene ikke bare er viktige i seg selv, men at det også er viktig å kunne gjøre «translations among them and transformations within them». Det finnes ulike versjoner av modellen til Lesh et al. (1987), men jeg velger å legge frem den versjonen som blir presentert hos NCTM (2014, s. 25), som er vist nedenfor i figur 1.



Figur 1. Ulike typer representasjoner. Hentet fra NCTM (2014, s. 25).

Denne modellen viser de fem ulike matematiske representasjonsformene som vanligvis blir omtalt i matematikk: *visuell, symbolsk, verbal, kontekstuell og fysisk.* Pilene mellom de ulike boblene symboliserer at det skal kunne gjøres translasjoner mellom de ulike representasjonsformene.

2.1.1 Semiotiske representasjoner

Duval (2006) snakker om begrepet semiotikk og mener at ingen matematiske prosesser finner sted uten bruk av semiotiske representasjoner. Semiotikk er læren av tegn og innebærer hvordan mennesker bruker tegn for å kommunisere (Cobley, 2001). I representasjonssammenheng handler det om hvordan representasjoner brukes for å skape mening (Curtin, 2009).

En klassisk beskrivelse av begrepet «representasjon» er ifølge Duval (2006, s. 103) at *en representasjon er noe som står for noe annet*. Samtidig mener han at denne beskrivelsen blir for flyktig, fordi representasjoner inneholder så mye mer. Representasjoner kan være et individs forståelser og misforståelser som en får tilgang til via individets verbale eller skjematiske produksjoner (Duval, 2006, s. 104). Samtidig kan det være de ulike komplekse forbindelsene mellom tegn, som sammen danner et system som er bygget opp av regler (Duval, 2006). Duval (2006, s. 104) sier at «the semiotic representations, including any language, appear as common tools for producing new knowledge and not only for communicating any particular mental representation». Her forteller han at semiotiske representasjoner fremstår som et verktøy for å produsere ny kunnskap, og ikke kun som et verktøy for å kommuniser en indre representasjon.

Duval (2006) er opptatt av hvordan man kan forstå de kognitive prosessene som skjer i arbeid med matematikk. Han skriver at «no kind of mathematical processing can be performed without using a semiotic system of representations, because mathematical processing always involves *substituting some semiotic representation for another*» (Duval, 2006, s. 107). En matematisk prosess innebærer ifølge Duval å bytte ut ulike former for semiotiske representasjoner med hverandre. Denne utbyttingen er en kognitiv prosess, og han beskriver disse prosessene som *transformasjoner.*

For at en transformasjon skal skje må vi først og fremst ha system av representasjoner som lar seg transformere. Disse representasjonssystemene kaller Duval (2006) for «representasjons-register» eller bare «register». Et register er altså et semiotisk system som tillater transformasjoner av de semiotiske representasjonene som inngår i dette systemet (Duval, 2006, s. 111). To eksempler på register er algebraiske notasjoner og verbalt språk. Når man utfører transformasjoner kan man enten gjøre dette innad i et register, eller ved å endre register. I og

med at dette er så forskjellige handlinger, gjør dette at Duval velger å skille mellom to typer transformasjoner: *treatments* og *conversions*, som jeg velger å oversette til *behandlinger* og *overganger*. Jeg vil nå se litt nærmere på disse.

Behandlinger er transformasjoner av representasjoner hvor representasjonene *ikke* endrer register (Duval, 2006, s. 111). Et eksempel her vil være i arbeid med grafer og diagrammer. En graf kan representeres som punkter i et koordinatsystem eller som søyler i et histogram. Selv om de to ser forskjellige ut, vil de være to ekvivalente representasjoner innenfor samme register.

Overganger derimot, er transformasjoner av representasjoner som innebærer en endring av register (Duval, 2006, s. 112). Et klassisk eksempel på en overgang er å gå fra et algebraisk uttrykk til en graf (eller andre veien). Algebraiske symboluttrykk som $f(x) = ax + b$ kan transformeres til en visuell linear graf ved bruk av for eksempel graftegneren GeoGebra. Her vil både grafen og det algebraiske uttrykket vise til det samme matematiske objektet, men med to ulike representasjonsformer. Dette bidrar til at overganger anses å være en mer kompleks transformasjon siden det innebærer å kunne gjenkjenne det samme matematiske objektet mellom to ulike representasjoner (Duval, 2006, s. 112).

Transformasjoner og register står sentralt i Duval (2006) sin teori om semiotiske representasjoner, og hvordan de kobles opp mot kognitive prosesser som foregår i læring av matematikk.

2.1.2 Hva er det som representeres?

Som blant annet Duval (2006) nevner, er en representasjon noe som står for noe annet. Når man snakker om matematiske representasjoner er det naturlig at disse skal representere noe matematisk. Men hva er det egentlig som representeres?

De ulike forfatterne jeg inkluderer i dette teorikapittelet benytter seg av forskjellige begreper når det gjelder hva som representeres. Eksempler som jeg velger å trekke frem er Duval (2006) som bruker begrepet «matematisk objekt», Goldin (2020) som bruker «matematisk idé eller sammenheng», Lesh et al. (1987) som bruker «matematisk idé», Arcavi (2003) som bruker «matematisk konsept og idé», og Sfard (1991) som bruker «matematisk konsept, objekt, idé, konstrukt, «notion» og «enitity»». Hvorfor de velger å benytte seg av de ulike begrepene kommer ikke tydelig til uttrykk hos de forskjellige.

Dette skaper rom for undring: Er det noe forskjell på for eksempel et matematisk *objekt*, *konsept*, *idé* og *sammenheng*? Her utelukker jeg ikke at det også kan finnes andre begreper som kunne passet inn i denne settingen. Begrepene påstår jeg er delvis vage, og det er vanskelig å

finne klare retningslinjer på hva som inngår i de enkelte. Sfard (1991) er derimot en av de som går mer i dybden på dette. Hun skriver om matematikkens dualitet, og trekker inn blant annet hvordan representasjoner bidrar til å uttrykke matematikkens verden:

> Unlike material objects, however, advanced mathematical constructs are totally inaccessible to our senses – they can only be seen with our mind's eyes. Indeed, even when we draw a function or write down a number, we are very careful to emphasize that the sign on the paper is but one among many possible representations of some abstract entity, which by itself can be neither seen nor touched. (Sfard, 1991, s. 3)

Ved videre undersøkelser viser det seg at å finne gode svar på dette krever et dypere dykk inn i matematisk filosofi, og Godino (1996) nevner at forståelsen rundt matematiske abstraksjoner må støttes av teori tilknyttet naturen av disse. Grunnet min oppgaves omfang og fokusområde velger jeg å ikke gå nærmere inn i den filosofiske tematikk bak matematikkens natur. Det er derimot et viktig aspekt som må inkluderes for å male et større bilde rundt matematiske representasjoner.

Når jeg i denne oppgaven ser på hva som representeres velger jeg å ta utgangspunkt i noe som er en felles enighet hos de ulike forfatterne: matematikk er *abstrakt.* Jeg gjør derfor her et valg om å forholde meg til at representasjoner representerer en eller annen abstrakt enhet (min oversettelse av Sfards «some abstract entity»), om det så er et matematisk objekt, konsept, idé, sammenheng etc. Min påstand er derfor at disse ulike begrepene er ulike sider av samme sak når vi snakker om representasjoner. Det begrepet jeg velger å benytte meg av videre i oppgaven er «matematisk objekt». Altså at matematiske representasjoner representerer et matematisk objekt hvor begrepet «matematisk objekt» vil fungere som et samlebegrep for matematisk konsept, idé, sammenheng og så videre.

For å gi en kort beskrivelse av matematisk objekter viser jeg til Duval (2006). Han legger frem en distinksjon mellom matematiske objekter og fenomener i de andre naturvitenskapene som blant annet astronomi, fysikk, kjemi og biologi. Matematiske objekter kan ikke måles med mikroskop, teleskop, metermål eller andre form for måle-instrumenter. Den eneste måten man får tilgang til matematiske objekter på er ved å bruke tegn og semiotiske representasjoner (Duval, 2006, s. 107).

2.1.3 Representasjoner i undervisningssammenheng

«National Council of Mathematics», videre omtalt som NCTM, inkluderer representasjoner som et element innenfor effektiv matematikkundervisning. Her påpekes det at læreren må

kunne engasjere elevene til å skape koblinger mellom ulike matematiske representasjoner for å kunne gi dem en dypere forståelse innenfor matematiske konsept samtidig som de kan bruke det som et verktøy i problemløsningsoppgaver (NCTM, 2014). Som vist tidligere i figur 1, henvises det vanligvis til fem typer representasjonsformer: *visuell, symbolsk, verbal, kontekstuell og fysisk.* Figuren viser også piler mellom disse representasjonen, som symboliserer at det skal kunne gjøres translasjoner mellom de ulike representasjonsformene. Begrepet translasjoner brukes av Lesh et al. (1987) som påpeker at «not only are these types of representation systems important in their own rights, but translations among them, and transformations within them, also are important» (Lesh et al., 1987, s. 2). Som vi ser nevnes det at transformasjoner innad i en representasjonsform også er viktig. Disse konseptene ser vi også kommer til uttrykk hos Duval (2006) sine overganger og behandlinger. Ifølge NCTM er disse translasjonene viktige med tanke på matematisk dybdeforståelse: «The depth of understanding is related to the strength of connections among mathematical representations that students have internalized» (NCTM, 2014, s. 25). Fra sitatet ser vi at dybdeforståelsen er avhengig av koblinger mellom representasjoner som studentene har internalisert. Internalisert betyr i denne forstand at ytre prosesser blir gjort om til indre prosesser. Et eksempel kan være at de ytre symbolene «$f(x) = ax + b$», først må prosesseres internt, før man videre kan overføre det til den verbale representasjonsformen og si «en lineær funksjon».

I 2020 startet innrullingen av den nye læreplanen LK20 her i Norge. En ting som er nytt med denne lærerplanen er at alle fag har fått tildelt noen grunnleggende *kjerneelementer*. Kjerneelementer er det viktigste faglige innholdet elevene skal arbeide med i opplæringen (Udir, 2019a). Udir skriver også at kjerneelementene er det elevene *må* (min kursiv) lære for å kunne mestre og å anvende faget. Innenfor matematikk R2 finnes det seks ulike kjerneelementer. Jeg vil her presentere det kjerneelementet som tar for seg representasjoner og kommunikasjon:

> Representasjon og kommunikasjon: Representasjoner i matematikk R er måter å uttrykke matematiske begreper, sammenhenger og problemer på. Representasjoner kan være konkrete, kontekstuelle, visuelle, verbale og symbolske. Det handler også om å forklare og begrunne valg av representasjonsform. Videre handler det om å oversette mellom matematiske representasjoner og språket i andre kontekster og om å veksle mellom ulike representasjoner. Kommunikasjon i matematikk R handler om å bruke matematisk språk i samtaler, argumentasjon og resonnementer. (Udir, u.å.a)

Først og fremst ser vi at Udir skriver at representasjoner er måter å uttrykke matematiske begreper, sammenhenger og problemer på. Disse vil inngå i samlebegrepet «matematiske objekter» som jeg la frem i forrige delkapittel. I utdraget ser vi også at Udir, på samme måte som NCTM (2014), presenterer fem ulike representasjonstyper. Disse fem er tilnærmet identiske, bortsett fra der Udir bruker begrepet «konkrete», bruker NCTM «fysisk». Disse to begrepene er synonymer av hverandre, og kan derfor sees på som samme type representasjon. Både hos Udir og NCTM kommer det til uttrykk at det å kunne veksle mellom ulike representasjoner er noe som inngår i det å bruke matematiske representasjoner.

2.1.4 Den verbale representasjonen

En sentral representasjonsform som kommer til uttrykk hos blant annet Udir (u.å.a) og NCTM (2014), er den verbale representasjonsformen. Matematiske verbale representasjoner innebærer ord og setninger som kan formuleres både skriftlig og/eller muntlig for å beskrive matematiske objekter. Det vil si at når man skriver eller sier noe for å beskrive et matematisk objekt, så bruker man den verbale representasjonsformen.

Wallace (2004, s. 906) bruker begrepet «to verbalize», som jeg oversetter til «å verbalisere». Her sier han at når vi skal uttrykke noe vitenskapelig til et nytt publikum, så må man gjøre om fra et vitenskapelig språk til et hverdagslig språk. Denne omgjøringen mellom to ulike typer språk krever en kognitiv tankeprosess som Wallace (2004) kaller for «å verbalisere». Hana (2013, s. 161) skriver også om å verbalisere, og sier at for å forstå betydningen av en matematisk notasjon, så kan det være lurt å venne seg til hvordan man uttrykker notasjonen muntlig. Dette kan igjen bidra til å si noe om at den verbale representasjonen kan styrke forståelsen av matematiske objekter, basert på den kognitive prosesseringen som skjer når vi skal formulere oss verbalt.

2.1.5 Den visuelle representasjonen

Av de fem representasjonstypene som blir presentert hos NCTM (2014) påpekes det at den visuelle representasjonsformen er spesielt viktig i matematikkundervisning. Arcavi (2003) forteller også om de ulike rollene visuelle representasjoner har i læring av matematikk. I artikkelen drøfter han begrepet *visualisering.* Innledningsvis trekker han frem en interessant digresjon; *Hva med blinde matematikere?* Arcavi ønsker å poengtere at visualisering handler om mer enn det man kan se med det blotte øyet. Dette beskriver han også som «seeing the unseen». Arcavi gir en definisjon av visualisering:

> Visualization is the ability, the process and the product of creation, interpretation, use of and reflection upon pictures, images, diagrams, in our minds, on paper or with technological tools, with the purpose of depicting and communicating information, thinking about and developing previously unknown ideas and advancing understandings. (Arcavi, 2003, s. 217)

Det å kunne visualisere handler i denne forstand om å kunne danne seg forestillinger mentalt, på papir eller ved teknologiske verktøy. Hensikten skal være å kommunisere informasjon, utvikle tidligere ideer og avansere vår forståelse.

Matematikken skiller seg ut fra de andre realfagene når det kommer til visualisering. I de andre realfagene har det som før har vært «usynlig» for mennesker blitt synlig ved hjelp av teknologi. Biologi, kjemi og fysikk er eksempler på områder hvor dette har skjedd. Ved hjelp av mikroskoper kan vi nå *se* celler og atomer – noe som tidligere eksisterte som en idé. Matematikkens abstrakte natur gjør at ingen optiske teknologier kan hjelpe oss å faktisk se de matematiske objektene (Arcavi, 2003, s. 216). Det vi har behov for er det Arcavi (2003) kaller for «kognitiv teknologi»: et medium som hjelper oss å overskride sinnets begrensninger når det kommer til tenking, læring og problemløsningsaktiviteter (Arcavi, 2003, s. 216). Matematisk visualisering kommer også til uttrykk hos Sfard (1991, s. 3) som sier at matematiske objekt kun kan bli sett med «our mind's eyes».

2.1.6 Den fysiske eller manipulerbare representasjonen

En tredje representasjonsform jeg vil trekke inn i dette kapittelet er den fysiske eller manipulerbare representasjonsformen. Denne formen for representasjon har ulike navn hos ulike forfattere. I modellen NCTM (2014, s. 25) brukes begrepet «physical», som kan oversettes til «fysisk». Udir (u.å.a) bruker begrepet «konkrete». Lesh et al. (1987, s. 2) bruker begrepet «manipulative models» i deres representasjonsmodell, som kan oversettes til «manipulative modeller». Til slutt vil jeg vise til Sarama og Clements (2009, s. 1) som skriver om «concrete manipulatives», som kan oversettes til «konkrete manipulasjoner». I modellen til NCTM (2014, s. 25) bruker de begrepet «fysisk», hvor de referer til å bruke fysiske objekter for å representere matematiske prosesser. Samtidig viser NCTM (2014, s. 84) til Sarama og Clements (2009) som foreslår at virtuelle manipulasjoner kan være ekvivalent til å bruke fysiske objekter. Det kan derfor diskuteres om begrepet «physical» i modellen til NCTM er dekkende nok for alle representasjoner.

Konkrete manipulasjoner kan foregå både fysisk, mentalt og virtuelt (Sarama & Clements, 2009, 2016). Sarama og Clements (2009, s. 146) bruker begrepet konkreter, men sier også at

det er viktig at man definerer hva man mener med dette begrepet. Fysiske konkreter kan for eksempel være brikker som brukes for å representere heltall. Elever kan eksempelvis gjøre manipulasjoner ved å legge til eller fjerne brikker, for å lære om addering og subtrahering. Mentale konkreter knytter Sarama og Clements (2016, s. 75) opp mot flinke elever som ofte kan manipulere størrelser mentalt, som om de skulle vært fysiske konkreter. Her forteller de videre at elevers oppfatning av konkrete manipulasjoner kan støtte opp de mentale, eller indre, representasjonene (Sarama & Clements, 2016, s. 75). «Konkreter» kan altså være både fysiske objekter, men som også kan oppfattes mentalt. «Konkreter» kan også finnes i form av virtuelle manipulasjoner.

Når det kommer til virtuelle manipulasjoner sier Sarama og Clements (2009, s. 147) at: «computers may provide representations that are just as personally meaningful to students as physical objects». Virtuelle manipulasjoner skjer altså på en datamaskin. Dette danner videre et spørsmål om hvorvidt det finnes «konkreter» i dette virtuelle formatet. Vanligvis vil begrepet «konkreter» henvise til fysiske objekter, men Sarama og Clements (2016, s. 75) skriver at «concrete may not equal physical». Herfra kan det tenkes at konkreter kan finnes virtuelt, noe som muliggjør virtuelle manipulasjoner.

Jeg vil også trekke inn en nyere artikkel fra Goldenberg et al. (2021) som rapporterer om pågående forskning tilknyttet blant annet programmering som noe virtuelt manipulerbart. De undersøker barneskoleelever som arbeider med matematiske oppgaver i det blokkbaserte programmeringsprogrammet «Snap!». I artikkelen blir det gjort sammenligninger mellom blokkene i «Snap!» og fysiske base 10-blokker. For eksempel sier Goldenberg et al. (2021, s. 62) at programmeringsblokkene er mer fleksible enn fysiske blokkene når det kommer til negative tallverdier. Goldenberg et al. (2021, s. 63) påpeker at observasjonene som blir lagt frem i rapporten er tentative, da dette er forskning som fremdeles er pågående. Siden artikkelen også har relevans inn mot programmering og matematiske representasjoner vil jeg se nærmere på den i delkapittel 2.3.2.

2.2 Programmering

I dette delkapittelet vil jeg fremlegger relevant teori tilknyttet programmering. Jeg vil starte med å se på programmering i skolen. Deretter vil jeg se kort på algoritmisk tenkning da dette er et begrep som har en sterk tilknytning til programmering. Videre vil jeg inkludere et delkapittel om programmeringsobjekter. Jeg vil også legge frem et par avsnitt hvor jeg ser på programmeringsspråk, før jeg til slutt ser kort på didaktiske programmeringsbarrierer.

2.2.1 Programmering i skolen

Freiman (2020, s. 870) forteller at de første datamaskinene sakte begynte å ta del av matematikkundervisningen mellom årene fra 1950 til 1980. På 70- og 80-tallet ble det designet programmer som krevde bruk av sitt eget programmeringsspråk for å fungere. Og for å få programmet til å operere så effektivt som mulig var det nødvendig å kunne språket (Freiman, 2020, s. 871). Et av disse språkene, LISP, ble videre benyttet for å utvikle et programmeringsspråk som var spesiallaget for matematikkundervisning (Freiman, 2020, s. 871). Dette språket heter LOGO og ble laget av blant annet Seymour Papert på 1980 tallet (Freiman, 2020, s. 871). Ideen til Papert var at programmering skulle fungere som en ressurs for å forbedre elevene i blant annet problemløsning, samtidig som det skilte seg fra de tradisjonelle «eksempel-oppgave-prøve» gjennomgangene fra klasserommet (Johnston-Wilder & Pimm, 2004; Sevik, 2016).

Fra år 1980 til år 2023 har bruken av teknologi økt drastisk. Flere og flere områder av våre hverdagsliv blir nå digitalisert, og klasserommene er intet unntak. Her i Norge har bruk av digitale verktøy blitt en integrert del av undervisningen på skolene. Et digitalt «verktøy» som markerte seg i årsskiftet 2022-2023 var den kunstige-intelligens-chatboten «chatGPT», som gikk viralt over hele verden. En NRK-artikkel skrevet av Eriksen (2022) viser til hvordan denne chatboten har skapt fortvilelse hos lærere, blant annet tilknyttet elevjuks. I artikkelen kommer det også til uttrykk at enkelte anser denne teknologiutviklingen som truende mot befolkningens kunnskapsutvikling. Samtidig gir det rom for å belyse viktigheten av programmeringskunnskap i matematikkfaget for å kunne bidra til å forstå slike programmer som «chatGPT». Munthe (2022b, s. 12) beskriver det slik: «kombinasjonen av matematisk kunnskap og programmeringskunnskap blir ansett som en verdifull ferdighet – ikke nødvendigvis det å kunne lage komplekse programmer, men kunnskapen om hvordan et program fungerer og dets muligheter og begrensninger» (min oversettelse).

Programmering har nå etablert seg i den norske skole. I matematikk har programmering fått en viktig plass. Kristensen og Kirfel (2022, s. 7) nevner at i utarbeidingen av de nye læreplanene var det matematikkfaget som fikk et ansvar for å gi elevene kompetanse i programmering og algoritmisk tenkning. I kompetansemålene til matematikk R2 nevnes bruk av programmering eksplisitt i to punkter (Udir, u.å.b):

- *Utforske rekursive sammenhenger ved å bruke programmering og presentere egne framgangsmåter*

- *Utvikle algoritmer for å beregne integraler numerisk, og bruke programmering til å utføre algoritmene*

Selve begrepet «programmering» blir belyst hos Sevik (2016). Hun nevner hvordan begrepet tidligere har blitt brukt for å omtale verbet «å skrive programkode». Dette har utviklet seg videre til begrepet «koding» som ofte blir benyttet i vår dagligtale. Jeg velger derimot å forholde meg til verbet «å skrive programkode». Sevik (2016, s. 9) velger derimot å se på programmering som mer enn bare det å skrive programkode: «det inkluderer også prosessen med å komme fram til denne koden. Det vil si prosessen fra å indentifisere et problem og tenke ut mulige løsninger på problemet, til å skrive kode som kan forstås av en datamaskin, og å feilsøke og kontinuerlig forbedre denne koden». Fra dette sitatet belyser Sevik et viktig poeng som jeg tar med meg videre i denne oppgaven. Poenget er at det å skrive programkode er noe som inngår i programmeringsprosessen. Programmering kan derfor anses som et overordnet begrep som innebærer ulike prosesser, blant annet det å lage strategi og å skrive en programkode.

Når man skriver programkode så skriver man på et spesifikt programmeringsspråk. Som et siste punkt i dette delkapittelet vil jeg nevne at det er programmeringsspråket «Python» som benyttes i matematikkfaget på videregående skoler i Norge (Munthe, 2022b, s. 36). Kristensen og Kirfel (2022, s. 9) legger frem fem grunner til hvorfor de velger å bruke Python:

1. Det er et språk som er lett å komme i gang med.
2. Det er mange moduler du kan bruke i Python.
3. Det er i dag mange universiteter som bruker Python for sine nye studenter.
4. Lærebøkene for videregående skole har valgt å bruke Python.
5. Python er et gratis programmeringsspråk med åpen kildekode som er tilgjengelig for alle å bruke.

Punkt 4 gjør at det også blir naturlig for lærere å benytte seg av Python når de har programmering i matematikkundervisningen. I min informantklasse er det også Python som har blitt benyttet som programmeringsspråk.

2.2.2 Algoritmisk tenkning

Jeg vil inkludere et delkapittel om algoritmisk tenkning, da jeg mener det er relevant satt opp mot programmering. Udir presenterer ikke en egen definisjon av begrepet programmering. Ser vi derimot på kjerneelementene «utforsking og problemløsing» hos Udir (u.å.a), blir vi introdusert for et annet begrep som har en sterk tilknytning til programmering. Dette begrepet

er «algoritmisk tenkning» (Udir, u.å.a). «Algoritmisk tenkning» er den norske oversettelsen av det engelske begrepet «computational thinking» (videre forkortet som CT) (Udir, 2019b).

Bråting og Kilhamn (2021, s. 172) skriver at begrepet CT først ble nevnt av Seymour Papert i 1996, den samme personen som lagde LOGO. De forteller videre at Papert introduserte begrepet på en tid hvor digitale verktøy ikke var helt integrert i hverdagslivene til folk, noe som igjen kan ha medført at Papert sine ideer ikke fikk stort gjennomslag. I dag er det annerledes og CT har fått sin egen plass i utdanningsløpet flere steder i verden (Bråting & Kilhamn, 2021). Selv om begrepet har blitt mye omtalt innenfor matematikkdidaktikken mangles det fremdeles en helt generell definisjon av begrepet «computational thinking» (Stephens & Kadijevich, 2020).

Begrepet «algoritmisk tenkning» er et begrep som har skapt diskusjon blant didaktikere. Mye av grunnen til dette er at «algoritmisk tenkning» er Udir (2019b) sin oversettelse for det engelske begrepet «computational thinking». Bakgrunnen for kritikken er at det norske begrepet ikke er like dekkende som det engelske (Wahlstrøm, u.å.). Videre i oppgaven velger jeg å forholde meg til begrepet «algoritmisk tenking», sammen med de beskrivelsene som er gitt hos Udir (2019b):

> 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. I dette ligger også en forståelse av hva slags problemer/oppgaver som kan løses med teknologi og hva som bør overlates til mennesker. (….) Det handler også om å lage abstraksjoner og modeller av den virkelige verden ved å fjerne unødvendige detaljer og fokusere på det som er relevant for den aktuelle problemstilling og løsning. En løsning på et spesifikt problem kan ofte generaliseres, slik at den kan brukes til å løse lignende problemer, og løsninger på flere delproblemer kan kombineres for å løse mer komplekse problem. (Udir, 2019b)

Et hovedpunkt som jeg vil trekke ut fra dette sitatet er at algoritmisk tenkning innebærer «*å kunne bruke sin teknologiske kompetanse for å få en datamaskin til å løse (deler av) problemet*». Herfra er det rimelig å anta at programmering og algoritmisk tenkning har en sterk tilknytning til hverandre. Jeg vil også trekke ut at algoritmisk tenkning handler om å lage abstraksjoner av den virkelige verden. Her kan det å lage abstraksjoner knyttes opp til å bruke ulike representasjonsformer.

2.2.3 Programmeringsobjekter

Brennan og Resnick (2012) legger frem et rammeverk for CT, hvor de deler opp CT i tre kategorier: «computational concepts», «computational practices», og «computational perspectives». Av disse tre ønsker jeg å greie ut om den førstnevnte kategorien «computational concepts» som jeg velger å oversette til «programmeringsobjekter». Brennan og Resnick (2012, s. 3) forteller at når man arbeider med et programmeringsspråk så vil man støte på programmeringsobjekter som er vanlige i mange programmeringsspråk. De programmeringsobjektene som Brennan og Resnick (2012) legger frem er valgt ut med hensyn på det blokkbaserte programmeringsspråket «Scratch». Jeg har valgt ut de objektene som er mest relevante sett i henhold til Python. Disse programmeringsobjektene er: *Sequences*, *loops*, *conditionals*, *operators* og *data* (Brennan & Resnick, 2012, s. 3). Disse velger jeg å oversette til *rekkefølger*, *løkker*, *betingelser*, *operatorer* og *datastrukturer*. En oversikt over disse programmeringsobjektene blir presentert i tabell 1. I tabellen gir jeg en kort forklaring og viser til konkrete eksempler til hvert objekt. Forklaringene er delvis inspirert av Brennan og Resnick (2012) og eksemplene er selvproduserte.

Programmeringsobjekt	**Forklaring**	**Eksempel**
Rekkefølger	Rekkefølger handler om hvordan datamaskinen leser de instruksjonene som er gitt i programmet. Det å lage et program kan ses på som en oppskrift som datamaskinen skal følge. Da er det viktig å tenke på hvilken rekkefølge de ulike instruksene skal gjennomføres. I Python leses programkoden kronologisk, altså at den leser linje for linje.	`x = 1` `z = x + y` `y = 3` I dette eksempelet vil ikke programmet fungere. Den får et problem i linje 3 når den skal beregne $z = x + y$ siden den ikke vet hva y-verdien er. I figuren under er rekkefølgen endret og programmet vet nå hva både x og y er, og kan gjennomføre utregningen. `x = 1` `y = 3` `z = x + y`
Løkker	Løkker er en mekanisme for å gjennomføre samme rekkefølge flere ganger.	Å lage et program som gir deg teksten «Hei!» fem ganger kan gjøres på to måter: 1) ``` print("Hei!") print("Hei!") print("Hei!") print("Hei!") print("Hei!") ``` 2) ``` for i in range(5): print("Hei!") ```

		3) ``` i = 0 while i<5: print("Hei!") i+=1 ``` I eksempel 2) og 3) brukes en *for*- og *while-løkke* for å utføre «print» operasjonen fem ganger.
Betingelser	Betingelser brukes for å få programmet til å avgjøre hvilke rekkefølger den skal følge. På denne måten kan ett program gi forskjellige resultater som avhenger av betingelsene.	``` x = 4 if x>10: print("OK") else: print("ikke OK") ``` I eksempelet brukes «if/else»-betingelsen for å sjekke om variabelen x er større enn tallet 10. Hva som blir resultatet av dette programmet, avhenger av hvilken verdi variabelen x har. I dette tilfellet vil programmet gi ut teksten «ikke OK» siden variabelen x har verdien 4 som *ikke* er større enn 10.
Operatorer	En operator er et uttrykk som får programmet til å utføre en matematisk operasjon. I dette programmeringsobjektet inkluderer jeg også innebygde funksjoner og nøkkelord som finnes i Python.	Operator: $**$ ``` x = 3**2 ``` Denne operatorer tilsvarer den algebraiske notasjonen: 3^2. I eksempelet er altså $x = 3 * 3 = 9$. Operator: sum() ``` verdier = [1,2,3] total = sum(verdier) ``` Denne operatoren kan brukes til å summere sammen tall i en liste. I eksempelet adderes tallene i listen «verdier», slik at «total» får verdien $1 + 2 + 3$ som er 6.
Datastrukturer	Datastrukturer handler om å lagre, gjenfinne og oppdatere verdier i programmet.	I Python kan verdier tildeles en bestemt datatype som for eksempel *str* (tekst), *float* (desimaltall) *og int* (heltall). Eksempler på ulike datastrukturer i Python kan være variabler, lister og dictionaries. ``` verdier = [int(5.5), float(5.5), str(5.5)] verdier.append(int(4.9)) ``` I eksempelet *er* «verdier» en liste med ulike elementer. I linje 3 brukes operatoren «append()» for å legge til heltallet 4.9 inn i listen. Figuren under viser hva som blir datastrukturen til «verdier» når programmet kjøres. ``` In [8]: verdier Out[8]: [5, 5.5, '5.5', 4] ```

Tabell 1. Programmeringsobjekter med forklaring og eksempel.

Programmeringsobjektene jeg har presentert i denne tabellen er sentrale i arbeid med programmering, og vil bli brukt videre i oppgaven.

2.2.4 Programmeringsspråk

Som nevnt tidligere er programmeringsspråket Python det som blir brukt i videregående skoler her i Norge. Python er et tekstbasert programmeringsspråk som bruker symboler fra tegnsettet ASCII. ASCII står for «American Standard Code for Information Interchange» (Loshin, u.å.), og innebærer blant annet alfanumeriske-, aritmetiske- og tegnsettingssymboler. Enkelt forklart er dette symboler vi vanligvis bruker når vi skriver med et tastatur. Når disse symbolene blir skrevet inn i programmeringsprogrammet Python vil de bli oversatt via noe som heter «UTF-8 encoding». Dette er et system som oversetter «vanlig» tekst til binære tall som kan leses av prosessorer i PC-en (Gazoni, 2018, s. 97).

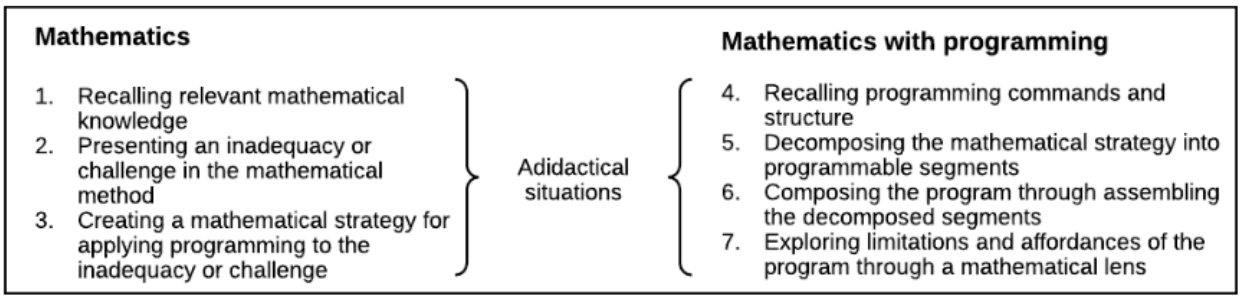
Gazoni (2018) skriver om semiotikk knyttet til programmeringsspråk. Han sier at den viktigste semiotiske forskjellen mellom programmeringsspråk og naturlig språk er knyttet til vaghet (Gazoni, 2018, s. 98). Menneskers naturlige språk har den egenskapen at det er vagt (Gazoni, 2018). For eksempel kan ordet «tre» bety enten tallet «3» eller planten «tre». Når man programmerer er det til syvende og sist datamaskinens prosessor som skal lese programmet som en har laget. Hvis du skriver «tre» til en datamaskin, så kan ikke den vurdere om du henviser til tallet eller gjenstanden. Derfor er det viktig at man er presis med hva man uttrykker når man skal formulere det som et programmeringsspråk, slik at det kan leses av en datamaskin. Denne presise språkbruken gjør at det ikke alltid er like lett å tyde hva et program uttrykker. Til dette påpeker (Gazoni, 2018) viktigheten bak det å skrive «program-dokumentasjon» med naturlig språk. I Python kan slik program-dokumentasjon skrives inn i selve programkoden ved å bruke symbolet «#».

Når man lager variabler i Python kan disse navnsettes slik man ønsker. Dette er ikke noe som vil påvirke oversettelsen til binære tall. Dette gir oss muligheten til å designe programkoden slik at den blir enklere lese for oss mennesker. Kristensen og Kirfel (2022, s. 15) nevner at når vi jobber med variabler i Python så kan det være lurt å bruke navn som er forklarende. Når vi for eksempel skal ha en variabel som inneholder en verdi for farten i oppgaven, så kan vi navngi denne variabelen for «fart» istedenfor å kalle den x. Når man skal angi navn på variabler i Python er det et par hensyn man må ta: Navnet må starte med en bokstav eller understrek, navn er versalsensitive (skiller mellom små og store bokstaver) og navn kan ikke være Python-nøkkelord som for eksempel *sum*, *for* eller *in* (Kristensen & Kirfel, 2022, s. 16). Det å bruke

hensiktsmessig navnsetting av variabler i Python er ingen nødvendighet for å lage et program som fungerer. Beskrivende navn på variabler kan derimot gjøre det lettere for andre å forstå programkoden. Dette kan igjen påvirke hvordan programkoden opptrer som en ekstern representasjon slik det blir beskrevet hos Goldin (2020).

2.2.5 «Mathematical programming problems» og programmeringsbarrierer

Både det å lære programmering, og det å lære programmering som en del av matematikkundervisningen, består av signifikante hinder og barrierer (Munthe, 2022a, s. 2). I artikkelen undersøker Munthe (2022a) hvilke hinder elevene møter på når de arbeider med matematisk programmering. For å gjøre dette har han opparbeidet en struktur som han kaller for «Mathematical programming problems (MPP)». MPP handler om å lage et program som løser et matematisk problem. I stedet for å gi elevene en MPP av formen «lag et program som …», velger Munthe å bruke et design som deler oppgaven opp i mindre deler. Denne oppdelingen blir presentert i figur 2.



Figur 2. Steg som brukes i et «mathematical programming problem». Hentet fra Munthe (2022a, s. 3).

Her ser vi at steg 1-3 handler om å knytte inn matematikken i problemet, og steg 4-7 handler om å knytte inn programmering. I arbeid med programmeringsoppgaver støter elevene på flere barrierer. Munthe (2022a, s. 6) viser til tre slike barrierer: «selection barriers», «understanding barriers» og «information barriers». Selection barriers handler om forståelsen av brukergrensesnittet rundt programmeringsapplikasjonen, og om kjennskap til verktøy og kommandoer som benyttes i arbeid med programmering. Understanding barriers handler om kunnskap knyttet til å løse feilmeldinger og uventet oppførsel fra programmet. Information barriers handler om å kunne evaluere om programmet løste det som skulle løses.

2.3 Programmering og matematiske representasjoner

Det er begrenset med tidligere forskning som ser på temaet programmering som en matematisk representasjon. I dette delkapittelet vil jeg derimot presentere tre artikler som jeg anser som

relevant for min problemstilling. Jeg vil først presentere to nyere artikler fra Bråting og Kilhamn (2021) og Goldenberg et al. (2021). Deretter vil jeg presentere en eldre artikkel fra Sfard (1991) som også belyser representasjoner og programmering.

2.3.1 Programmering og algebraiske notasjoner

I artikkelen «Exploring the intersection of algebraic and computational thinking» utforsker Bråting og Kilhamn (2021) hvordan programmeringsrelaterte representasjoner samhandler med tradisjonelle algebraiske representasjoner. Et av forskningsspørsmålene de tar for seg omhandler likheter og ulikheter mellom hvordan variabler, likhetstegn, funksjoner og algoritmer blir representert i dataprogrammer versus algebraiske notasjoner.

Bråting og Kilhamn (2021, s. 171) sier at programmeringsspråk kan bli sett på som et nytt representasjonssystem som kan benyttes for å representere matematiske objekter og strukturer. Samtidig sier de også at programmeringsspråkets hovedfunksjon er å forenkle dataprosessering og kalkulasjoner, noe som kan medføre at objektene som blir presentert ikke alltid er matematiske, og at tenkningen ikke alltid er algebraisk.

I artikkelen presenterer Bråting og Kilhamn (2021) tre eksempler fra tre ulike programmeringsprogram fra tre ulike skoletrinn. De to første eksemplene tar for seg programmene Lightbot (programmeringsspill) og Scratch (blokkbasert programmeringsspråk). Jeg velger derimot å se nærmere på det tredje eksempelet, hvor programmeringsspråket «JavaScript» blir brukt. Dette gjør jeg fordi JavaScript har flere likhetstrekk med Python i og med at de begge er tekstbaserte programmeringsspråk. I eksempelet blir det lagt vekt på hvordan symbolet «=» brukes i både programmering og algebra, men representere ulike ideer. I algebra beskriver «=» en ekvivalensrelasjon, mens i programmering brukes symbolet for å tildele en verdi til en variabel (Bråting & Kilhamn, 2021, s. 180). Her er det også viktig å trekke frem at en variabel i programmering ikke er det samme som en variabel i matematikken, selv om vi bruker det samme begrepet. Kristensen og Kirfel (2022, s. 14) eksemplifiserer dette med følgende programkode:

```
1  a = 2
2  b = 4
3  print(a+b)
```

Til denne programkoden sier Kristensen og Kirfel (2022, s. 14) at selv om a og b kalles for variabler, så referer de her til bestemte tall, eller verdier, hvor a er tallet 2, og b er tallet 4. De varierer altså ikke i størrelse, på samme måte som variabelen x i funksjonsuttrykket $f(x) =$

$x^2 - 3$ (Kristensen & Kirfel, 2022). Jeg velger å benytte meg av variabler for slik elementer som a og b i det presenterte eksempelet.

Tilbake til Bråting og Kilhamn (2021), hvor de viser til en linje med følgende programkode: «$a = a + 1$». Fra et algebraisk perspektiv kan uttrykket leses som «a er lik a pluss 1», noe som vil være en matematisk umulighet da det for eksempel impliserer at $1 = 2$. Fra et programmeringsperspektiv kan uttrykket leses som «adder 1 til verdien a», noe som innebærer å tildele en ny verdi til variabelen a, som er en helt vanlig operasjon i programmering. Uttrykket «$a = a + 1$» kan også omskrives til «a+= 1» som vil gjennomføre tilsvarende operasjon. Eksempelet belyser hvordan likhetstegnet «=» og andre tegn som for eksempel «%» har ulike betydninger i programmeringsspråk og algebraiske notasjoner. Dette påvirker også betydningen av disse tegnene når de omtales med naturlig språk, hvor overgangen mellom naturlig språk til JavaScript (programmering) er forskjellig fra overgangen til algebraiske notasjoner (Bråting & Kilhamn, 2021, s. 180).

I den avsluttende diskusjonen til Bråting og Kilhamn (2021, s. 181-182) beskriver de programmering sin representasjonsrolle. De sier at selv om programmeringsspråk har sin opprinnelse fra algebra så danner et programmeringsmiljø nye register med syntaktiske regler som skiller seg fra naturlig språk og algebraiske notasjoner. De nevner også at det å gjøre overganger fra naturlig språk til et annet register, for eksempel «en variabel» i et programmeringsspråk, krever kunnskap om hvilket objekt ordet «variabel» referer til. Naturlig språk skiller seg fra programmeringsspråk på grunn av sin vaghet (Bråting & Kilhamn, 2021, s. 174). Denne vagheten innebærer at ord og begreper fra et naturlig språk er avhengig av hvilken kontekst man befinner seg i, mens poenget med et programmeringsspråk er å være presist uten noe form for vaghet (Bråting & Kilhamn, 2021).

2.3.2 Programmering og virtuelle manipulasjoner

I delkapittelet om den fysiske representasjonsformen så jeg på hvordan manipulasjoner også kunne gjøres virtuelt på en datamaskin. Her trakk jeg inn artikkelen til Goldenberg et al. (2021) som undersøker programmering som noe virtuelt manipulerbart. De har undersøkt barneskoleelever som arbeider med matematikk via det blokkbaserte programmeringsspråket «Snap!».

I artikkelen presenteres det ulike funn og observasjoner. Samtidig påpeker Goldenberg et al. (2021, s. 63) at observasjonen de presenterer er tentative, da datainnsamlingsprosessen fremdeles er pågående. Artikkelen ble publisert i februar 2021, og jeg har ikke klart å finne en

artikkel med oppdatert forskning. Jeg antar derfor at forskningen fremdeles er pågående. I dette delkapittelet vil jeg legge frem noen av de observasjonene som blir lagt frem i artikkelen. Jeg vil understreke at studien til Goldenberg et al. (2021) undersøker barneskoleelever som arbeider med et blokkbasert programmeringsprogram, og at jeg i min studie undersøker videregående elever som arbeider med et tekstbasert programmeringsspråk. Det kan derfor diskuteres i hvor stor grad deres observasjoner vil være overførbare til min studie. Jeg vil uansett presentere artikkelen da jeg mener den inneholder elementer som er relevante for min studie.

Goldenberg et al. (2021, s. 49) rapporterer først og fremst programmering og manipulasjoner. Men de presenterer også fire observasjoner, som de ikke var forberedt på å finne da de startet forskningen. Diss fire observasjonene er knyttet til (1) abstraksjon, (2) presis kommunikasjon, (3) problemløsning og (4) indre representasjoner. Jeg vil legge frem observasjon (1), (2) og (4) da jeg anser disse som mest relevant for min studie.

Observasjonen om abstraksjon handler om internalisering av manipulasjoner. Goldenberg et al. (2021) forteller her at elever har enklere for å danne mentale abstraksjoner når de driver med programmeringsblokker kontra når de driver med fysiske objekter. Dette ble observert når de spurte elevene om å forutsi hva outputen til programmet kom til å være. Goldenberg et al. (2021, s. 62) sier at «the abstraction already seemed to be in their heads». De forteller videre at det er uvisst hvorfor det er slik.

Observasjonen om presis kommunikasjon handler om at programmering gjorde at elevene ble tydeligere i sine verbale uttrykk av matematiske handlinger. Dette begrunner Goldenberg et al. (2021, s. 49) med at elevene har en lettere tilgang til de matematiske prosessene, eller algoritmene, når de allerede har gitt instrukser til datamaskinen. Når elevene da blir bedt om å forklare så har de allerede sorterte tankene sine, noe som gjør det lettere å gi en forklaring. Dette står i kontrast til hva som blir sett i elevers arbeid med fysiske manipulasjoner.

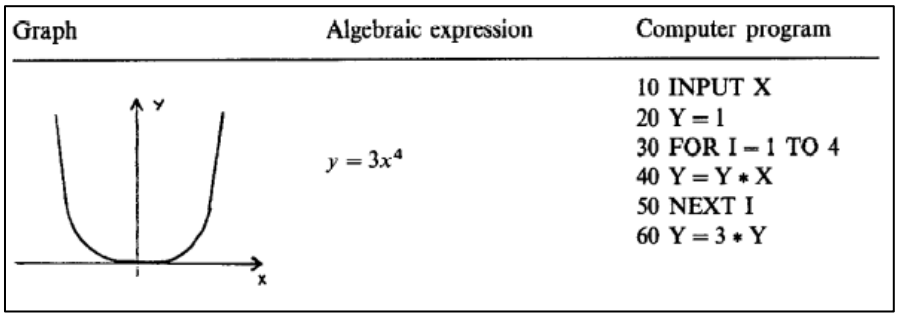
Observasjonen om indre representasjoner handler om at programmering kan bidra til å danne bedre indre representasjoner av matematiske objekter. Goldenberg et al. (2021, s. 50) gir her et eksempel på at det å lære å programmer algoritmer knyttet til manipulasjoner av brøk, kan bidra til å gjøre det lettere å få tilgang til det matematiske objektet. De forteller videre at dette synes å være et tilfelle, men at det foreløpig er spekulasjoner.

Disse tre observasjonen som jeg nå har lagt frem kan bidra til å si noe om programmering og matematiske operasjoner. Observasjon (1) handler om manipulasjoner som inngår i den «fysiske» representasjonsformen. Observasjon (2) handler om elevenes verbale

representasjoner som er en sentral matematisk representasjonsform. Til slutt har vi observasjon (3) hvor det spekuleres i om programmering bidrar til å danne indre representasjoner som gir en enklere tilgang til matematiske objekter. I og med at forskningen til Goldenberg et al. (2021) er pågående, blir det spennende å se hva de finner ut videre.

2.3.3 Strukturell og operasjonell forståelse

Sfard (1991) skriver om matematikkens dualitet hvor hun inkluderer matematiske representasjoner. I artikkelen gjør hun også koblinger mellom representasjoner og programmering. Sfard (1991, s. 4) introduserer to ulike forståelser av matematiske objekter: *strukturell* og *operasjonell* forståelse. Hun beskriver den strukturelle forståelsen som «static, instantaneous and integrative», som jeg oversetter til statisk, umiddelbart og helhetlig. Imens den operasjonelle er «dynamic, sequential and detailed» som jeg oversetter til dynamisk, sekvensiell og detaljert (Sfard, 1991, s. 4). I figur 3 nedenfor bruker Sfard det matematiske objektet *funksjon* for å eksemplifisere den strukturelle og operasjonelle forståelsen.



Figur 3. Ulike representasjoner av en funksjon. Hentet fra Sfard (1991, s. 6).

Figuren viser til hvordan en funksjon kan beskrives ved hjelp av ulike representasjoner - som en graf, et algebraisk uttrykk eller som programkode. Den grafiske fremstillingen gir oss muligheten til å begripe funksjonen som en helhet ved at mange punkter fra funksjonen har blitt kombinert til å forme en kontinuerlig strek. Den grafiske fremstillingen oppfordrer derfor til en strukturell forståelse (Sfard, 1991, s. 6). Det algebraiske uttrykket kan derimot forstås på begge måter ifølge Sfard (1991, s. 6) som sier: «Den kan forstås operasjonelt som en beskrivelse av en utregning, eller strukturelt som et statisk forhold mellom to mengder» (min oversettelse). Til slutt har vi programkoden som det kan se ut til at korresponderer til en operasjonell forståelse da den presenterer funksjonen som en algoritmisk prosess (Sfard, 1991, s. 6).

Sfard (1991) forteller videre at den strukturelle og operasjonelle forståelsen ofte viser seg i de ulike representasjonene vi bruker når vi prosesserer kunnskap kognitivt. Dette innebærer blant annet hvordan vi visualiserer matematiske objekter ved hjelp av indre representasjoner. Noen

indre representasjoner kan være strukturelle, mens andre kan være operasjonelle (Sfard, 1991, s. 7). I tillegg forteller hun at verbale uttrykk må bli prosessert sekvensielt, og er derfor mer passende til å representere algoritmiske prosesser.

3.0 Forskningsdesign og metode

I dette kapittelet vil jeg legge frem prosjektets forskningsdesign. Kapittelet har som formål å fortelle *hva* jeg har gjort for å undersøke problemstillingen, og *hvorfor* jeg har gjort det. Jeg vil starte med å presentere forskningsdesignet. Deretter vil jeg presentere og begrunne valg av metoder for datainnsamling, før jeg videre forteller om selve datainnsamlingsprosessen. Beskrivelse av analyse og analyseprosessen vil også blir presentert. Mot slutten av kapittelet vil jeg diskutere studiens kvalitet, før jeg til slutt reflekterer over etiske perspektiv for studien.

3.1 Et kvalitativt forskningsdesign

For å beskrive forskningen jeg har utført i sammenheng med denne masteroppgaven benytter jeg meg blant annet av begrepene forskningsdesign og metode. Forskningsdesign er ifølge Gleiss og Sæther (2021, s. 25) en plan for hvordan undersøkelsen skal gjennomføres. I denne planen skal undersøkelsen avgrenses ved å spesifisere problemstilling, forskningsmetode, utvalg og teori. Disse delene av forskningsdesignet vil igjen påvirke hvilke funn og konklusjoner man kommer frem til (Gleiss & Sæther, 2021, s. 26).

For dette prosjektet har jeg valgt å bruke et kvalitativt forskningsdesign i form av det jeg har valgt å kalle for et hermeneutisk-fenomenologisk casestudie. Som metode har jeg brukt lyd-, skjerm- og video-opptak fra elevenes PC-er.

Kvalitativ forskning

Hos Gleiss og Sæther (2021, s. 29) blir begrepene «kvantitativ og kvalitativ metode» omtalt som «upresis begrepsbruk». Det som derimot skaper et skille mellom disse to er om kategoriene som forskeren bruker til å analysere datamaterialet er fastlagt på forhånd eller ikke (Gleiss & Sæther, 2021, s. 30). Bryman (2012, s. 36) forteller at det er vanskelig å sette en strek som skiller mellom kvantitative og kvalitative forskningsstrategier, men at det finnes kontraster som skaper en distinksjon mellom dem. En av disse kontrastene er at den kvantitative strategien *tester* teori, og at den kvalitative *generer* teori. Grunnet begrenset teori som var tilknyttet til akkurat min problemstilling kunne jeg ikke ta utgangspunkt i teori jeg kunne teste, og jeg kunne derfor heller ikke fastlegge noen bestemte kategorier på forhånd av analyseprosessen. Derfor var jeg også interessert i å få et rikt bilde av elever i en programmeringssituasjon. Postholm og Jacobsen (2018, s. 95) forteller at beskrivelse, forståelse og mening er sentrale begreper i en kvalitativ studie. For å undersøke problemstillingen min mest mulig åpent valgte jeg derfor å bruke et kvalitativt forskningsdesign.

Hermeneutisk-fenomenologisk casestudie

Tre forskningsdesign har blitt vurdert som passende for å bidra til å svare på problemstillingen min: casestudie, fenomenologisk studie og hermeneutisk studie. Jeg har valgt å kalle denne kombinasjonen for en hermeneutisk-fenomenologisk casestudie.

En casestudie klassifiseres av studien er begrenset til én analyseenhet (Grønmo, 2016, s. 105). I og med at jeg samlet inn data fra én undervisningsøkt med én VG3-klasse kan studien min sies å være en casestudie. Samtidig sier Grønmo (2016, s. 105) at et av formålene med casestudier kan være å utvikle en helhetsforståelse av akkurat denne enheten som studeres. I bunn og grunn var det tilfeldigheter som gjorde at jeg endte opp med å undersøke akkurat denne klassen, og mitt fokus for problemstillingen er ikke å undersøke akkurat denne VG3 klassen. Casestudien gir meg derimot muligheten til å undersøke programmering som en matematisk representasjon i en spesifikk klasseromskontekst.

Mitt fokus var å undersøke programmering som en matematisk representasjon, og det er her den fenomenologiske studien er viktig å trekke inn. En fenomenologisk tilnærming i et kvalitativt design handler om å utforske og beskrive mennesker og deres erfaringer med og forståelse av et fenomen (Christoffersen & Johannessen, 2012, s. 99). Når det kommer til hermeneutisk studie forteller Grønmo (2016, s. 394) at hermeneutikk innebærer at forskeren skal fortolke handlingene til elevene sett i lys av konteksten de inngår i og den situasjonen eller prosessen som handlingen er en del av.

Disse tre designene kombineres til det jeg kaller for en hermeneutisk-fenomenologisk casestudie. Jeg vil ikke fastlåse meg til én av disse tilnærmingene da jeg påstår at alle bidrar til å avdekke perspektiver knyttet opp mot undersøkelsen av problemstillingen *programmering som en matematisk representasjon*. Felles for alle tre designene er at de appellerer til et kvalitativt forskningsdesign.

3.2 Kontekst og deltakere

Jeg vet at programmeringskompetansen er forskjellig fra klasserom til klasserom. Dette kommer som en følge av at læreres kompetanse også er forskjellig, i og med at programmering fortsatt er nytt i matematikkundervisningen. På bakgrunn av dette satte jeg meg et kriterium om at informantklassen skulle ha noe erfaring med programmering. Grunnen til dette var for å unngå mest mulig av det Munthe (2022a, s. 2) kaller for *programmeringsbarrierer*. Dette er hinder elever møter på når de arbeider med programmering. Av de ulike barrierene han legger frem var jeg mest opptatt av å unngå det han kaller for «selection barriers» (Munthe, 2022a, s.

6). Denne barrieren er knyttet til forståelsen av brukergrensesnittet rundt programmeringsapplikasjonen, og om kjennskap til verktøy og kommandoer som benyttes i arbeid med programmering. Utvalget som ble gjort på bakgrunn av dette kan derfor klassifiseres som det Gleiss og Sæther (2021, s. 39) kaller for et *ikke-sannsynlighetsutvalg*, eller *kriteriebasert utvalg*, hvor forskeren velger deltakere basert på forhåndsbestemte kriterier.

Jeg kom i kontakt med en lærer som hadde en klasse i matematikk R2, hvor elevene hadde arbeid litt med programmering tidligere. Denne læreren vil videre bli omtalt som «faglæreren». I og med at disse elevene hadde noe programmeringserfaring, og faglæreren var åpen for å la meg gjennomføre forskningen, var informantklassen funnet. Videre måtte jeg også få godkjenning fra elevene. Det var frivillig for alle elevene å delta, og det ble utdelt samtykkeerklæringer på forhånd av datainnsamlingen. Dette blir forklart grundigere i delkapittel 3.7. Utvalget endte til slutt opp med å bestå av 24 elever. Mitt forskningsprosjekt er derfor basert på empiri hentet fra en R2-matematikklasse på Vestlandet.

3.3 Metode for datainnsamlingen

Valg av metode er noe som inngår i Gleiss og Sæther (2021, s. 28) sin forklaring av forskningsdesign. Hos (Bryman, 2012, s. 46) beskrives forskningsmetode som en teknikk for å samle inn data. Problemstillingen min omhandler programkode som matematisk representasjon. Som nevnt hos Goldin (2020) kan representasjoner være både interne og eksterne, og av disse to så er det de eksterne representasjonene som er tilgjengelig for å bli observert av andre. Samtidig sier Goldin (2020, s. 566) at representasjoner er noe som blir produsert. Jeg ville derfor gjennomføre en metode som kunne gi meg tilgang til elevers eksterne produksjoner av språk og programkode.

3.3.1 Lyd-, skjerm- og videoopptak

I og med at jeg ville undersøke hva som ble sagt av elevene var det naturlig at jeg ville bruke lydopptak som metode. Samtidig var jeg interessert i å fange opp hva elevene produserte på PC-en når de arbeidet med programmeringsoppgavene. Det var også viktig for meg at lydopptakene og skjermopptakene skulle være synkronisert slik at jeg kunne observere hva som ble sagt samtidig som eleven skrev programmet. Løsningene ble å bruke møte-funksjonen i programmet «Microsoft Teams», som ga meg muligheten til å få én fil med synkronisert skjerm- og lydopptak. Denne løsningen ga meg også muligheten til å inkludere videoopptak fra webkamera som en metode for datainnsamlingen. Bruk av videoopptak ville bidra til å gi meg en så nyansert situasjonsforståelse som mulig.

Postholm og Jacobsen (2018, s. 131) forteller at i observasjon kan lyd- og video-opptak være en god hjelp for forskeren. Når slike metoder skal brukes er det viktig at forskeren gjør seg kjent med det utstyret som skal brukes (Postholm & Jacobsen, 2018, s. 131). Dette var noe jeg tok hensyn til i forberedelsen av datainnsamlingen hvor jeg og faglærer testet ut opptaksfunksjonen i Teams. Siden elevene skulle gjøre opptakene på sine egne PC-er visste jeg at det kom til å være variasjon i lyd og bildekvalitet. Totalt sett ble ikke dette noe problem, selv om det ble vanskelig å tyde alt som ble sagt på enkelte grupper.

Alle skjermopptakene ble av god kvalitet og det ble ikke noe problem å tyde hva elevene gjorde på PC-ene. Det kan være viktig å nevne at elevene fikk beskjed om å dele hele skjermen, og at de ikke skulle ta opptak av kun et vindu på PC-en. På denne måten ga skjermopptakene meg viktig innsikt i hvordan de ulike elevgruppene jobbet. Det kunne for eksempel være at det var helt stille på en gruppe hvor én elev satt og skrev programkode. Her fikk jeg da muligheten til å observere hva som blir skrevet, selv om ingenting blir sagt. Skjermopptakene ga meg også mulighet til å observere om de vekslet mellom å bruke ulike programmer, nettsider eller lignende under gjennomførelsen av oppgaven.

Videoopptak er positivt i den forstand at det gjør det mulig for forskeren å gjenoppleve situasjonene flere ganger. Samtidig kan videoopptak virke negativt da det kan oppleves forstyrrende og utfordrende for forskningsdeltakerne (Postholm & Jacobsen, 2018, s. 131). Jeg vil påstå at denne påvirkningen fra videoopptak vil være forskjellig om man bruker webkamera kontra et fysisk kamera. Elevene er vant til å arbeide på PC-en og å sitte foran et webkamera, noe som muligens bidro til å gjøre dem mindre bevisst på at de ble filmet. Bruk av videoopptak viste seg å bli en viktig del for analyseprosessen, spesielt i transkriberingsfasen. Video fra webkamera fanger opp et relativt smalt område, og gir derfor ikke et fullstendig bilde av alt som skjer på gruppene. Samtidig gjorde videoopptakene det mulig for meg å skille hvem som sa hva, på de ulike elevgruppene. Dette var spesielt en fordel på grupper som hadde elever med like stemmer. Det gjorde det også mulig for meg å se blant annet om en elev skrev programkode alene, om de henviste seg til noe de hadde skrevet på papir, eller om andre de andre på gruppen tok frem sin egen PC.

3.3.2 Min rolle som forsker

Hvordan jeg valgte å opptre i rollen som forsker var noe jeg måtte ta hensyn til i forberedelse og gjennomføringen av datainnsamlingen. I forberedelsene til datainnsamlingen måtte jeg ta spesielt hensyn til valg av felt, og avgjøre grad av åpenhet, som er to av flere punkter som nevnes hos Grønmo (2016, s. 156). Under gjennomføring av datainnsamlingen var både jeg og

faglærer aktivt til stede i klasserommet. På denne måten ble jeg som forsker både en deltaker og observatør (Grønmo, 2016, s. 155). Dette var et bevisst valg da jeg ville kunne ha muligheten til å interagere med de ulike elevgruppene for å observere hva de gjorde underveis i økten. Valg av felt kan i min situasjon oversettes til valg av klasserom hvor jeg ville ha et klasserom som var stort nok til at gruppene kunne sitte spredt, samtidig som jeg kunne bevege meg rundt å observere alle. Dette hjalp faglærer med. Grad av åpenhet innebærer at jeg som forsker må ta hensyn til hvor mye informasjon om observasjon og studien som skal gis til aktørene (Grønmo, 2016, s. 158). Både i forberedelsene og under datainnsamlingen var jeg alltid åpen om hva som skulle observeres.

Gleiss og Sæther (2021, s. 111) påpeker at i en datainnsamlingssituasjon vil det være vanskelig å unngå påvirkning fra forskeren, og at spørsmålet ikke handler om hvorvidt forskeren påvirker, men hvordan forskeren påvirker. Grønmo (2016, s. 158) nevner blant annet at aksept, tillit og utvikling av feltrelasjoner mellom forsker og aktører er en av hovedoppgavene for gjennomføring av datainnsamling. Jeg følte at jeg fikk en god feltrelasjon til min informantklasse, mye fordi faglærer introduserte meg til klassen en stund før selve datainnsamlingen. Jeg fikk også være til stede i en undervisningsøkt hvor jeg benyttet anledningen til å bli kjent med elevene på forhånd av datainnsamlingen. I og med at datainnsamlingen tok del av en undervisningsøkt som omhandlet et tema elevene allerede arbeidet med, så sa lærer at det som ble gjennomgått ville være pensum og relevant til en eventuell vurderingssituasjon. Dette vil jeg påstå bidro til å flytte fokuset fra det som var forskningsrelatert, over til det som var skolerelatert. Selv om jeg hadde rolle som forsker i klasserommet, følte jeg meg ikke malplassert. Likevel må jeg anta at min tilstedeværelse og oppførsel kan ha påvirket elevene i en viss grad.

3.4 Datainnsamlingsprosessen

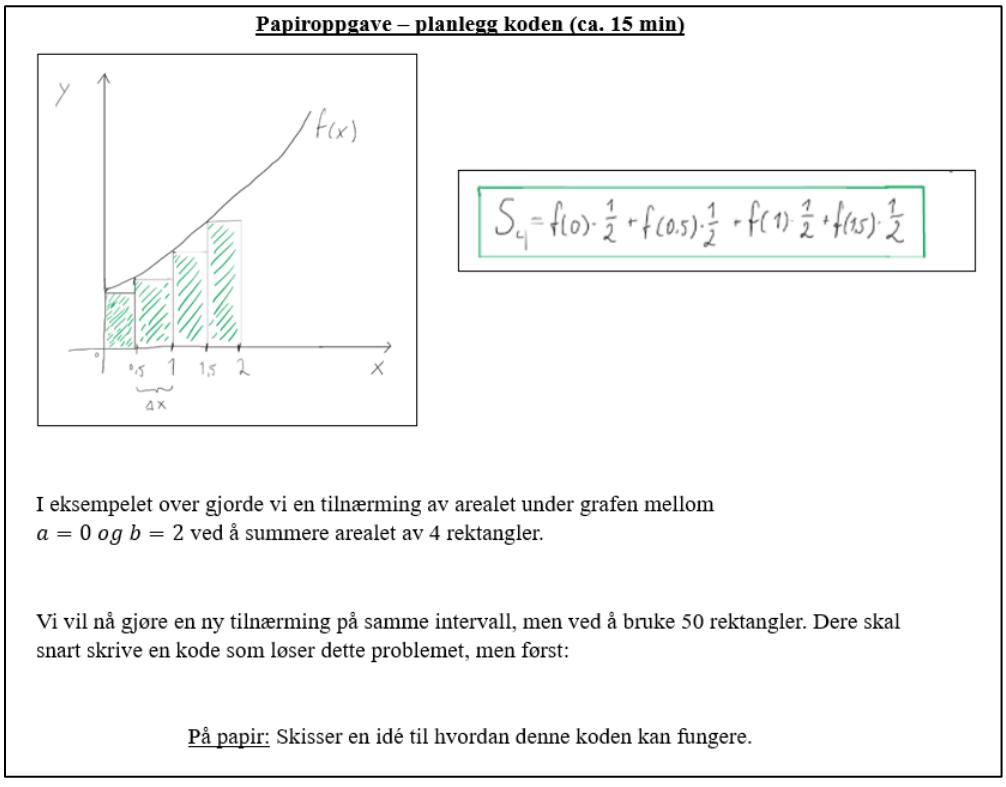
I dette delkapittelet vil jeg fortelle om ulike stadier for innhenting av data. Dette innebærer stadiene før, under og etter datainnsamlingen.

3.4.1 Før datainnsamlingen – forberedelser

Etter at utvalget av informanter var gjort, kunne jeg iverksette forberedelsene til selve datainnsamlingsdagen. For å gjøre datainnsamlingen ble jeg og faglærer enig om å gjennomføre innsamlingen i løpet av én undervisningsøkt. Før jeg hadde kommet i kontakt med faglærer var jeg usikker på hvilket matematisk tema jeg skulle ta for meg for datainnsamlingen. Det viste seg at faglærer nylig startet på pensum om integrasjon og integrasjonsmetoder, hvor numerisk

integrasjon var et av de temaene han hadde planlagt å gjennomføre med klassen. Dette ble dermed en mulighet for meg å innføre datainnsamlingen inn i et undervisningsopplegg med numerisk integrasjon som tema.

En av tingene som måtte forberedes til datainnsamlingen var oppgaven som skulle gis til elevene. Faglærer viste meg hvilke programmeringsoppgaver elevene hadde arbeidet med tidligere, og ga meg et innblikk i nivået til informantklassen. Det viste seg at nivået i klassen var litt spredt da det var noen som hadde programmering som valgfag ved siden av matematikken. Jeg ville ikke gi en oppgave som var for vanskelig for elevene, slik at de ble sittende fast og ikke fikk til det de skulle. Derfor tenkte jeg det ville være hensiktsmessig å dele oppgaven inn i to deler. I del 1 ville jeg at elevgruppene skulle ha en idémyldring innad i gruppen. Dette endte opp med å bli det jeg kaller for «papiroppgaven», som blir vist i figur 4 nedenfor.

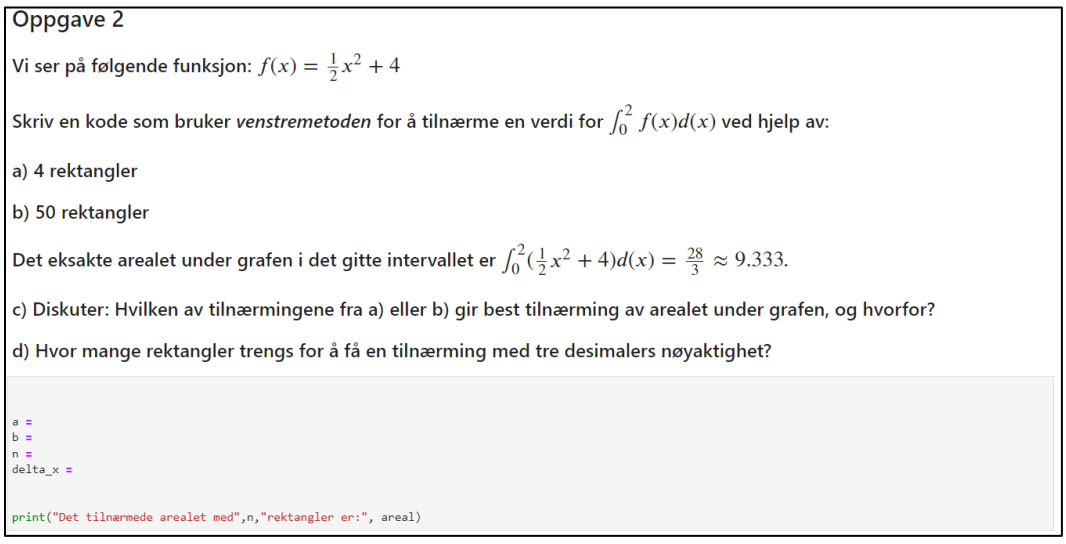


Figur 4. Egenprodusert papiroppgave som ble gitt til elevene.

Del 1 besto derfor av oppgaven som blir vist ovenfor. Denne oppgaven ble utdelt på papir, og gikk ut på at elevene skulle skissere og planlegge koden på et utdelt A4 ark, uten å bruke datamaskin. Arkene kunne jeg samle inn når økten var ferdig og bruke som datamateriale. Elevene fikk ca. 15 minutter på papiroppgaven.

I del 2 ville jeg at elevene skulle arbeide med en programmeringsoppgave på PC. Denne delen besto av fem oppgaver med tilhørende deloppgaver. I oppgave 1 skulle elevene skrive et program som beregnet bredden Δx på intervallet $[a, b]$. I oppgave 2 skulle elevene lage et program som tilnærmet arealet under en graf ved å bruke venstremetoden. I oppgave 3 skulle elevene lage et program som tilnærmet arealet under en graf ved å bruke høyremetoden. Både venstre- og høyremetoden handler om å summere sammen rektangler, men hvor rektanglene har ulik høyde. I oppgave 4 skulle elevene lage et program som tilnærmet arealet under en graf ved å bruke trapesmetoden, som handler om å summe sammen trapeser. I oppgave 5 skulle elevene diskutere en visuell fremstilling av numerisk integrasjon hvor de kunne velge antall rektangler i et program som plottet rektangler under en graf. Alle oppgavene med tilhørende deloppgaver og figurer er lagt ved i vedlegg 1.

Del 2 ble utlevert via nett-applikasjonen «JupyterLab». Dette er et program med et brukergrensesnitt som gir tilgang til egne tekstbokser hvor en kan skrive tekst, og egne programmeringsbokser, hvor en kan skrive programkode. JupyterLab er praktisk i en undervisningssammenheng fordi læreren kan gi oppgaver i tekstboksene, og så kan elevene skrive programkode i boksen under. Figur 5 nedenfor viser eksempelvis hvordan oppgave 2 ble seende ut i JupyterLab.



Figur 5. Oppgave 2 i JupyterLab.

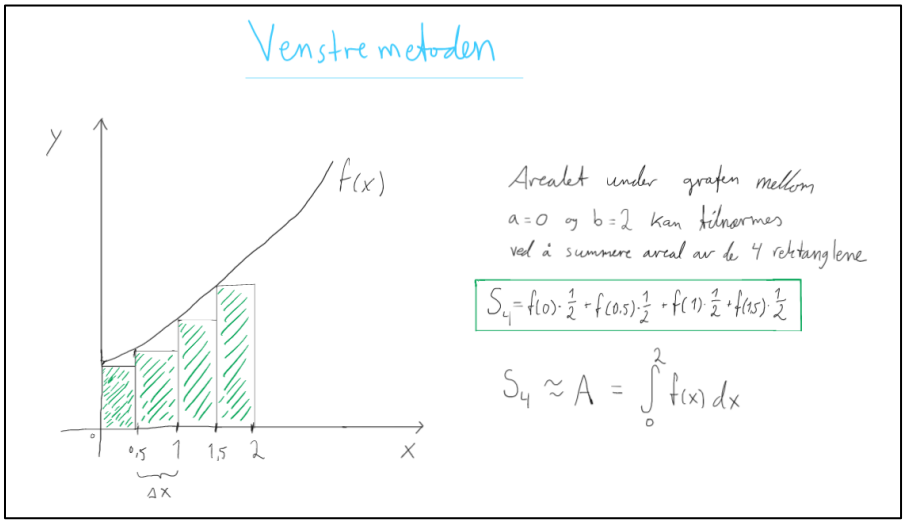
Den grå boksen i figuren viser til programmeringsboksen hvor elever kan skrive programkode som kan kjøres. Teksten over den gå boksen er oppgaveteksten. Denne applikasjonen gir altså elevene tilgang til å både lese oppgaven og skrive programkode samtidig. JupyterLab var også et program som faglærer hadde benyttet seg av i tidligere undervisningssammenhenger, og var

derfor noe informantelevene hadde kjennskap til. Programmeringsspråket som brukes i JupyterLab er Python.

3.4.2 Under datainnsamlingen – selve gjennomføringen

Når oppgaven til elevene var ferdig konstruert, og opplegget for undervisningstimen var ferdig planlagt, var både jeg og faglærer klar for å gjennomføre datainnsamlingen. Både planleggingen og gjennomføringen av datainnsamlingen ble gjort i samarbeid med faglæreren. Han hjalp blant annet med organisering av klasserom, gruppeinndeling og å sørge for elevene gjennomførte opptakene på korrekt måte. Faglæreren ble derfor en viktig del forskningsprosessen. Han ga meg adgang til skolen og opprettet kontakt mellom meg og informantklassen. Faglæreren ble derfor det Gleiss og Sæther (2021, s. 110) kaller for en *portvakt*: «En lærer som lar forskeren komme inn i skolemiljøet».

Jeg valgte som sagt å gjennomføre datainnsamlingen som en del av en undervisningsøkt med informantklassen. Timen startet med at jeg hadde en kort repetisjonsforelesning om integrasjon og hvordan integrasjon kan sees på som arealet under en graf. Deretter ga jeg en liten forklaring av numeriske metoder og introduserte elevene for «venstremetoden». Måten jeg presenterte dette på tavlen for elevene var tilnærmet lik det som er vist i figur 6.



Figur 6. Venstremetoden som presentert i oppgave 1, vedlegg 1.

Deretter ble klassen delt inn i 7 tilfeldig valgte grupper, med 3-4 elever på hver gruppe. Gruppene ble plassert spredt rundt i klasserommet for å unngå støy på lydopptakene fra nabogruppene. Når gruppene hadde kommet på plass forklarte faglærer hvordan de skulle sette opp et møte på Microsoft Teams for å kunne utføre opptakene som skulle gjøres. Her ble det også gitt beskjed til elevene at de kun skulle bruke én PC per gruppe. Dette ble gjort av to

grunner: (1) Da ville jeg få samlet opptakene fra én gruppe i ett opptak, og (2) for å øke den verbale aktiviteten i gruppen. Grunn (2) var inspirert av Liljedahl (2021) sitt konsept om «Building Thinking Classrooms», hvor han viser til at tilfeldige grupper med én tavle og én tusj per gruppe har en økt effekt på idé-bidrag fra elevene (Liljedahl, 2021, s. 46, 58). Nå benyttet ikke jeg meg av tavler og tusj i min situasjon, men jeg gjorde en antakelse om at det å bruke én PC og ett tastatur per gruppe muligens ville bidra til å gi den samme effekten av økt verbal aktivitet.

Når alle disse praktiske tiltak var ordnet kunne jeg starte å dele ut oppgavene til elevene, og be dem om å starte opptakene. Gruppene arbeidet deretter med oppgavene i omtrent 1 time og 30 minutter før undervisningsøkten ble avsluttet. Jeg avsluttet økten 10 minutter før den egentlig var ferdig slik at gruppene fikk tid til å lagre opptakene og sende dem inn til faglæreren.

3.4.3 Etter datainnsamlingen – datainnhenting

Innhenting av data var en relativt kort prosess. Når elevene stoppet opptakene, ble de opplastet direkte til skylagringssystemet til informantskolen. Herfra kunne faglærer laste ned opptakene, og deretter overføre opptakene over til mitt lagringssystem. I alle disse leddene befant opptakene seg på sikre lagringssteder, slik at etiske retningslinjer ble ivaretatt. Opplastning og nedlastningen av filene til skylagringssystemer førte derimot til at opptakene fra den ene gruppen ble utilgjengelige. Dermed satt jeg igjen med opptak fra 6 grupper, med 3-4 elever per gruppe.

3.5 Analyseprosessen

I dette delkapittelet vil jeg presentere analyseprosessen for min studie. Braun og Clarke (2006, s. 15) forteller at analyseprosessen begynner i det man starter å legge merke til, og å se etter, mønster eller elementer som interesserer med tanke på problemstillingen. Min analyseprosess startet derfor smått under selve gjennomføringen av datainnsamlingen, hvor jeg gjorde observasjoner som var interessante.

I og med at et kvalitativt datamateriale ofte er omfattende, gjelder det i analyseprosessen å få en oversikt over dette materialet slik at det kan presenteres for andre (Postholm & Jacobsen, 2018, s. 139). Jeg vil derfor fortelle om hvordan jeg har strukturert datamaterialet mitt for å kunne gjøre en analyse.

3.5.1 Analytisk tilnærming

Som nevnt innledningsvis fantes det lite tidligere forskning på akkurat programmering som en matematisk representasjon. Dette påvirket også hvordan jeg skulle bestemme min analytiske

tilnærming. Det finnes ulike metoder for hvordan man skal gjøre analyse av forskningen som har blitt gjennomført. Postholm og Jacobsen (2018, s. 157) presenterer for eksempel seks ulike analysevarianter: Analyse av casestudier, etnografiske analyse, fenomenologisk analyse, narrativ analyse, tekstanalyse og samtaleanalyse. Braun og Clarke (2006) skriver om tematisk analyse, som også er en kjent analysemetode.

Jeg har i min analyseprosess ikke fastlåst meg til én analysemetode. Mitt hovedfokus har vært å få et strukturert og godt innblikk i datamaterialet mitt, for å kunne undersøke problemstillingen og forskningsspørsmålene mine. I denne prosessen har jeg foretatt meg metoder som kan knyttes til deler av både analyse av casestudie og fenomenologisk analyse. Innenfor analyse av casestudie snakker Postholm og Jacobsen (2018, s. 157) blant annet om direkte analyse og kategorisk opphopning. I direkte analyse bygger forskeren forståelse på en enkeltstående situasjon, og i kategorisk opphopning bygger forståelsen på flere tilfeller, hendelser eller situasjoner (Postholm & Jacobsen, 2018, s. 157). Fenomenologisk analyse innebærer blant annet å skaffe et helhetsinntrykk av datamaterialet, og å lete etter interessante og sentrale temaer (Christoffersen & Johannessen, 2012, s. 100).

Som utgangspunkt for analyseprosessen min valgte jeg å basere analysen min på de to forskningsspørsmålene mine. Dette førte til en analytisk prosess som bar preg av både induktiv og deduktiv analyse. For å undersøke det første forskningsspørsmålet mitt, hvordan programkode kan betraktes som en matematisk representasjon, ville jeg ikke sette noen faste rammer for hvordan jeg skulle analysere datamaterialet. Dette blir ifølge Braun og Clarke (2006, s. 12) en induktiv analyse. Dette er en analysemetode som er «databestemt», hvor en ikke vil lete etter elementer som skal passe inn i forhåndsbestemte koder (Braun & Clarke, 2006, s. 12).

Analyseprosessen knyttet til det andre forskningsspørsmålet om hvordan programmering støtter opp under elevers verbale representasjon, kan minne om det Braun og Clarke (2006, s. 12) beskriver som en deduktiv, eller teoretisk, analyse. I en deduktiv analyse er mønstrene som skal identifiseres forhåndsbestemte, gjerne med utgangspunkt fra teori (Braun & Clarke, 2006, s. 12). For å undersøke det andre forskningsspørsmålet brukte jeg kategorier som var bestemt på forhånd, men som ikke var bestemt på teori. Kategoriene besto av matematiske objekter som jeg selv hadde funnet. Jeg vil derfor påpeke at denne delen av analyseprosessen kan anses som en deduktiv analyse.

Viktige deler av denne analyseprosessen var transkribering og strukturering av datamaterialet. Beskrivelser av hvordan dette har blitt utført blir fortalt i delkapitlene som følger under. Jeg vil også nevne at selv om jeg i analyseprosessen tok utgangpunkt i forskningsspørsmålene, så var også problemstillingen, programmering som en matematisk representasjon, reflektert opp mot det som ble analysert.

3.5.2 Transkribering

Videre vil jeg ta for meg transkriberingsprosessen av datamaterialet. Denne prosessen kan forstås som første steg i en mer systematisk analyseprosess (Gleiss & Sæther, 2021, s. 97). Etter å ha samlet inn alt datamaterialet, hadde jeg til sammen 9 timer med opptak som skulle transkriberes. Disse opptakene besto av både lyd-, skjerm- og videoopptak.

Transkribering handler om å overføre muntlige utsagn om til skriftlig tekst, og når man skal transkribere er det en del avgjørelser som må tas (Gleiss & Sæther, 2021, s. 98). Det som er viktig er at transkripsjonene er tilpasset til det de skal brukes til (Langdridge, 2006, s. 258). En avgjørelse jeg måtte ta var å bestemme hvor detaljerte transkripsjonene mine skulle være. Her bestemte jeg meg for å ta utaganspunkt i det Langdridge (2006, s. 259) kaller for «enkel transkripsjon». Dette er en transkripsjon som beskriver hvem som sa hva i kronologisk rekkefølge. Her transkriberte jeg direkte hva elevene sa, uten å trekke inn for mange detaljer rundt *hvordan* det ble sagt. I og med at jeg hadde tilgang til lyd- og skjermopptak valgte jeg å inkludere observasjoner fra disse i transkripsjonene. Dette gjorde jeg på steder hvor jeg følte et behov for å sette uttalene i en større kontekst. Selv om dette gjorde transkriberingsprosessen relativt tidkrevende, gav det til gjengjeld et datamateriale som inneholdt et nyansert bilde av elevenes kontekst og dialog.

En utfordring som møtte meg tidlig i transkriberingsprosessen var omforming av elevenes matematiske uttrykk. For eksempel ville det symbolske uttrykket «$s = f(i) \cdot \Delta x$» bli uttrykt verbalt som «*s er lik f av i ganger delta x*». Hana (2013, s. 161) skriver om verbalisering av matematiske notasjoner. Han forteller at når matematiske notasjoner bli verbalisert så kan det oppstå nyanser og misforståelser mellom lærer og elev, da de ikke nødvendigvis bruker ord i samme betydning (Hana, 2013, s. 165). Siden det kunne oppstå situasjoner hvor jeg var usikker på hvilken symbolsk formulering elevene snakket om, bestemte jeg meg for å transkribere de verbale uttrykkene med ord istedenfor symboler.

Langdridge (2006, s. 259) forteller om at man kan benytte seg av ulike transkriberingssystemer basert på hvilken type analyse som skal gjøres. Samtidig forteller han at det ikke finnes en fasit

på hvilke system som skal brukes, men at systemet må passe til formålet med arbeidet. Mitt transkriberingssystem ble henholdsvis enkelt. Hvis jeg ville legge til kontekst gjorde jeg dette i parenteser «( )». Her la jeg blant annet inn beskrivelser av hva elevene gjorde og hva som viste på skjermen. Hvis det var perioder hvor det enten var irrelevant dialog, eller en lengre pause, så skilte jeg med å bruke tre punktum «…». Den siste tegnsettingen min var hvis en elev ble avbrutt eller ikke fullførte en setning, hvor jeg da avsluttet elevens utsagn med to punktum «..».

3.5.3 Strukturering av datamaterialet

Hensikten med kvalitative analysemetoder er blant annet å sortere datamaterialet som er samlet inn, og gjøre det forståelig (Postholm & Jacobsen, 2018, s. 139). Etter jeg hadde transkribert ferdig datamaterialet satt jeg igjen med et Microsoft Word-dokument på over 100 sider og med over 30 000 ord.

Transkriberingsfasen var en viktig inngangsport for strukturingen av datamaterialet. Transkriberingen handlet ikke kun om å skrive ned elevenes muntlige tale. For å sette transkripsjonene mine i kontekst så jeg gjennom skjerm- og videoopptakene samtidig som transkriberte lydopptakene. Dette var en prosess som bidro til mange observasjoner som jeg tok med meg videre i strukturingen av datamaterialet. Når jeg var ferdig med all transkripsjon valgte jeg å gå mer systematisk til verks på å strukturere materialet mitt. Denne systematiske prosessen har jeg valgt å dele opp i tre deler.

Del 1 – Skrive kommentarer i margen

I den første delen gikk jeg gjennom alle mine transkriberte data og lagde kommentarer i margen underveis. Her hadde jeg allerede lagd noen kommentarer fra transkriberingsfasen. Alt dette ble gjort i programmet Microsoft Word. I denne prosessen hadde jeg induktiv tilnærming. Måten jeg analyserte på kan ses på som en kombinasjon av fenomenologisk analyse og analyse av casestudie, som ble beskrevet i delkapittel 3.5.1 om analytisk tilnærming. Etter hvert som jeg gikk gjennom transkripsjonene lagde jeg kommentarer på utsagn og hendelser som virket interessante for problemstillingen min. Mesteparten av disse hendelsene var enkelthendelser som foregikk på de ulike elevgruppene. Samtidig var det også hendelser som ikke var interessante i seg selv, men siden de dukket opp hos flere grupper så ble det lagt merke til. Dette kan ses på som det Postholm og Jacobsen (2018, s. 157) kaller for kategorisk opphopning. I og med at det ikke oppsto så veldig mange hendelser jeg anså som interessante, ble det ikke gjort noen spesifikk systematisering av disse. Med dette mener jeg at det ikke ble laget et eget system på koding og tematisering av innhold i transkripsjonene.

I denne delen av analysen noterte jeg altså hendelser og situasjoner som jeg oppfattet som interessant opp mot problemstillingen og forskningsspørsmålet. Enkelte av disse interessante hendelsene blir presentert som funn i kapittel 4.

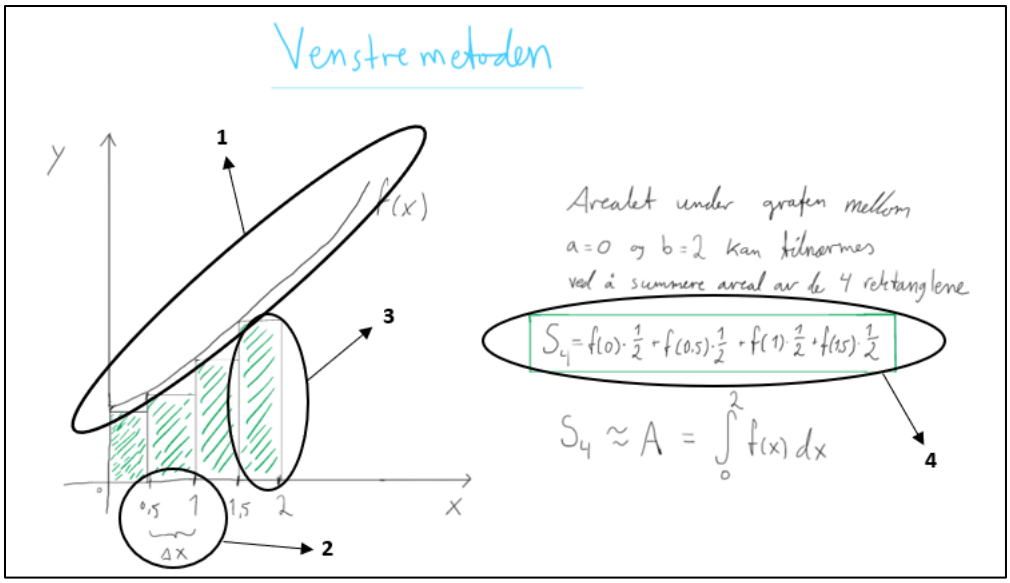
Del 2 – Matematiske objekter i elevers verbale representasjoner

I den andre delen av systematiseringen valgte jeg å sette fokuset over på forskningsspørsmål 2, om hvordan programmering kan støtte opp under elevers matematiske verbale representasjon. Før jeg iverksatte denne prosessen bestemte jeg meg for hva jeg skulle se etter i transkripsjonene. Derfor anser jeg denne delen av analyseprosessen som deduktiv. Som nevnt i teorikapittelet er det de *matematiske objektene* som representeres når vi snakker om matematiske representasjoner. Derfor var jeg interessert i å undersøke hvordan de matematiske objektene ble representert i elevdialogene. Før jeg begynte å analysere transkripsjonene ville jeg starte med å se etter hvilke matematiske objekter jeg kunne forvente å finne i arbeid med numerisk integrasjon. For å finne disse matematiske objektene undersøkte jeg numerisk integrasjon med venstremetoden, slik det blir representert i den symbolske og visuelle representasjonsformen. Jeg så også om jeg kunne finne de samme matematiske objektene i programkoden til elevene, slik at jeg visste at gruppen på et tidspunkt kan ha snakket om disse matematiske objektene. Deretter gikk jeg videre for å analysere de verbale representasjonene. De matematiske objektene som jeg identifiserte ved å undersøke den symbolske og visuelle representasjonsformen er presentert i tabell 2.

Nummer	**Matematisk objekt**
1	Funksjon
2	Bredde
3	Areal av rektangel
4	Sum av areal

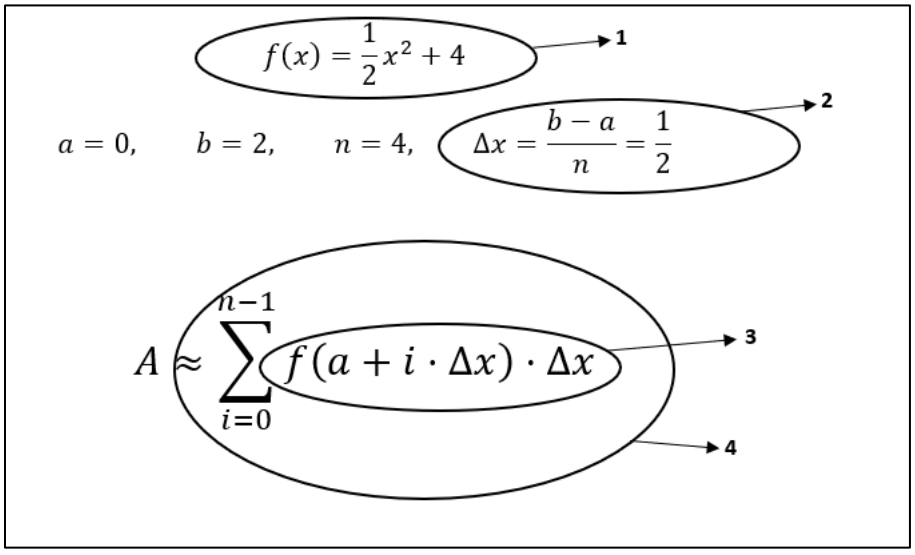
Tabell 2. Nummererte matematiske objekter.

De matematiske objektene presentert i denne tabellen ble som sagt identifisert ved å undersøke den symbolske og visuelle representasjonsformen av numerisk integrasjon. For å illustrere hvordan jeg identifiserte disse har jeg lagt ved to figurer, hvor jeg har satt en sirkel rundt det matematiske objektet som blir representert. Den første figuren viser, figur 7, viser en illustrasjon som ble gitt i oppgave 2, hvor elevene skulle tilnærme arealet med venstremetoden.



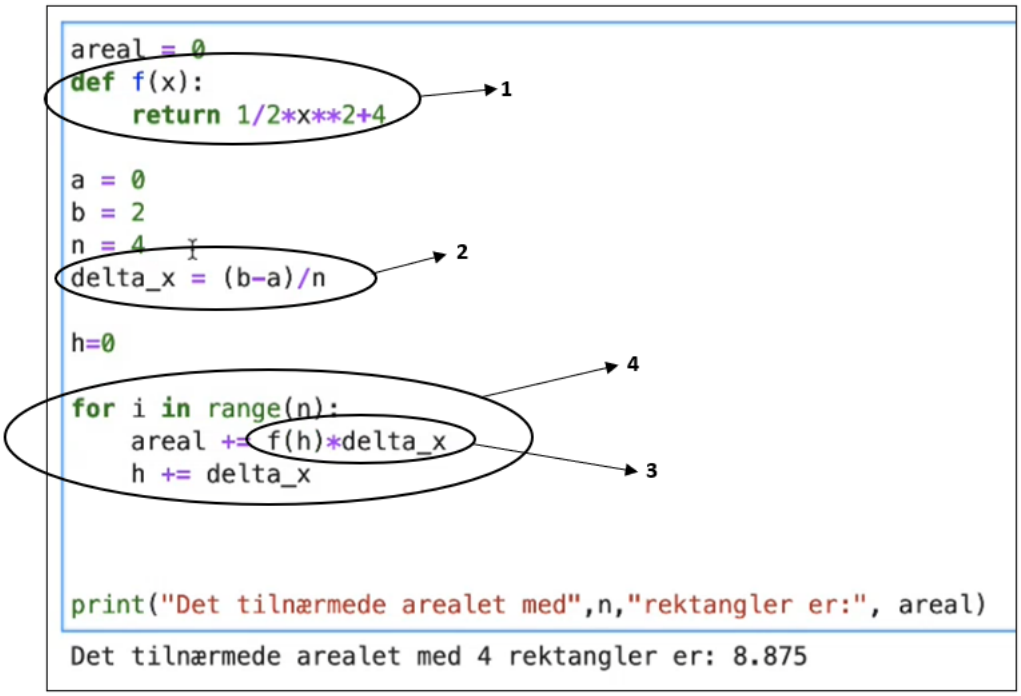
Figur 7. Illustrasjon gitt i oppgaven til elevene.

Figuren består hovedsakelig av visuelle representasjoner i form av graf og geometriske rektangler, men i illustrasjonen har jeg også valgt å inkludere symboler for å representere sum av areal (se sirkel 4). Figur 8 nedenfor, viser til den algebraiske formelen for numerisk integrasjon.



Figur 8. Algebraisk uttrykk av numerisk integrasjon.

Denne består kun av symbolske representasjoner. De ulike nummererte sirklene viser som sagt til de ulike matematiske objektene som ble presentert ovenfor i tabell 2. Deretter fant jeg de samme matematiske objektene i programkoden til elevene. Figur 9 viser programkoden til den ene elevgruppen, hvor de skulle tilnærme et areal ved bruk av venstremetoden.



Figur 9. Matematiske objekter uttrykt i programkoden til den ene elevgruppen.

Her er det meg selv som har vurdert hvilke deler av programkode som tilhører til hvilket matematisk objekt. Når disse fire matematiske objektene var identifisert, kunne jeg starte å se gjennom transkripsjonene. For å undersøke de verbale representasjonene laget jeg meg en tabell hvor jeg i den ene kolonnen hadde det matematiske objektet, og i den andre kolonnen la inn utdrag fra elevenes verbale representasjoner, markert her som «[utdrag fra elevdialog]». Oppsettet for tabellen er presentert i tabell 3.

Matematisk objekt	Verbal representasjon
Funksjon	[utdrag fra elevdialog]
Bredde	[utdrag fra elevdialog]
Areal av rektangel	[utdrag fra elevdialog]
Sum av areal	[utdrag fra elevdialog]
x-verdier	[utdrag fra elevdialog]

Tabell 3. Tabell for å systematisere de verbale representasjonene etter matematiske objekter.

Legg merke til at jeg i denne tabellen har inkluderte et femte matematisk objekt, nemlig *x-verdier*. Dette matematiske objektet var ikke noe jeg identifiserte ved å ta utgangspunkt i den visuelle og symbolske representasjonsformen. Når jeg startet å lese transkripsjonen la jeg merke til at det var en del dialog rundt x-verdier, noe som gjorde at jeg valgte å inkludere det som et matematisk objekt. Et annet sentralt matematisk objekt som burde nevnes er *grenseverdier*.

Grenseverdier er egentlig sentralt matematisk objekt i arbeid med numeriske metoder, men i transkripsjonene fant jeg lite verbal aktivitet rundt dette, noe som førte til at jeg ikke inkluderte det i tabellen.

Del 3 – Programmeringsobjekter og programmeringsbegreper i de verbale representasjonene

Etter å ha lagt inn utdrag fra elevdiskusjonene i en tabell, valgte jeg gjøre en videre analyse av de ulike utdragene. I tabellen hadde jeg inkludert elevdialoger hvor matematiske objekter kom til uttrykk i de verbale representasjonene. I forskningsspørsmålet vil jeg undersøke hvordan programmering støtter opp under disse verbale representasjonene, og jeg valgte derfor å lete etter utsagn i dialogen som var relevant mot programmering. Dette gjorde jeg ved å se etter programmeringsobjekter og programmeringsbegreper. Disse programmeringsobjektene presenterte jeg i delkapittel 2.2.3, og består av *rekkefølger*, *løkker*, *betingelser*, *operatorer* og *datastrukturer*. Hva som gjelder som programmeringsbegreper ble gjort som en underveisvurdering når jeg leste dialogene, og var derfor ikke forhåndsbestemte. Som oftest handlet dette om at elevene trakk inn begreper fra ulike Python-operatorer inn i dialogen sin.

De ulike struktureringsdelene jeg nå har lagt frem bidro til å gi meg en god oversikt over datamaterialet mitt. Dette gjorde det mulig for meg å sette meg inn i elevkonteksten, gjøre analyse og observasjoner, som igjen har endt opp med å bli funn for oppgaven.

3.6 Studiens kvalitet

Gleiss og Sæther (2021, s. 201) forteller at forskning hele tiden er gjenstand for vurdering og tilbakemeldinger hvor også forskeren selv har et ansvar på å vurdere og reflektere over kvaliteten på eget forskningsarbeid. Når forskningen skal evalueres er det ifølge Bryman (2012, s. 46) tre kriterier som er mest fremstående: reliabilitet, replikasjon og validitet. Replikasjon er et kriterium som sjeldent benyttes i samfunnsforskning, og når det først blir benyttet er det i tilknytning til kvantitativ forskning (Bryman, 2012, s. 47). Jeg vil derfor fokusere på å evaluere min forsknings reliabilitet og validitet. Innenfor validitet vil jeg se nærmere på begrepsvaliditet, indre validitet, ytre validitet og økologisk validitet.

Reliabilitet handler om i hvilken grad tilfeldige målingsfeil har påvirket dataene (Kleven & Hjardemaal, 2018, s. 99). Kleven og Hjardemaal (2018, s. 99) forteller at en studie som har god reliabilitet er lite påvirket av tilfeldige målingsfeil. For å vurdere reliabiliteten til studien min vil jeg se nærmere på to hovedaktører for studien: Elevene som informanter, og meg selv som forsker. Jeg vil starte med å vurdere i hvilken grad tilfeldige målingsfeil kan være relatert til elevenes dagsform under observasjonen. Her kan det stilles spørsmål om i hvilken grad

resultatet er avhengig av hvilke dager eller tidspunkt på dagen som observasjonen tilfeldigvis fant sted (Kleven & Hjardemaal, 2018, s. 101). For å vurdere dette forutsetter det at man har gjennomført samme måling på de samme personene, men ved ulike tidspunkter. Dette er noe jeg ikke har gjort i min studie, noe som gjør det vanskelig å vurdere det Kleven og Hjardemaal (2018, s. 101) kaller for «stabilitetsaspektet» ved reliabiliteten. Skulle jeg ha gjennomført en ny undersøkelse måtte jeg også ha tatt i betraktning at elevene har forandret seg ulikt noe som igjen gjør det vanskelig å si noe om reliabiliteten knyttet til informantelevene.

Videre vil jeg ta i betraktning tilfeldige målingsfeil tilknyttet meg som forsker. I denne studien er det kun meg som har gjort observasjoner og funn. Gleiss og Sæther (2021, s. 202) forteller at når forskningens reliabilitet skal vurderes må det tas hensyn til både utførelsen av datainnsamlingen og analysen av datamaterialet. For eksempel vil analysen av datamaterialet være påvirket av forskerens subjektive tilnærming noe som medfører bias (Gleiss & Sæther, 2021, s. 203). Bias er et annet ord for undersøkelseseffekter som refererer til feil og skjevheter i forskningen som kan påvirke resultatene. Man kan unngå bias ved å blant annet sjekke om funn er konsistente på tvers av ulike metoder, eller å samarbeide med andre forskere for å få en felles forståelse for koding av datamaterialet (Gleiss & Sæther, 2021, s. 203). Dette er grep jeg ikke har benyttet meg av i min forskning. Hadde jeg gjort slike grep ville det bidratt til å styrke studiens reliabilitet. Selv om det er vanskelig å unngå bias, har jeg i forskningsprosjektet mitt prøvd å opptre mest mulig objektivt for å styrke påliteligheten bak min empiri og mine funn. Dette innebærer blant annet å aktivt reflektere over hvordan jeg som forsker kan ha påvirket forskningsprosessen under innsamling av data, i analysen av data og i tolkning av data.

Validitet omtaler Bryman (2012, s. 47) som det viktigste kriteriet for forskning. Validitet handler om integriteten til de konklusjonene som blir gjort på grunnlag av studien (Bryman, 2012, s. 47). Kleven og Hjardemaal (2018) skriver om kritisk tolkning og vurdering av forskningsresultater. De legger frem tre typer validitet som jeg ønsker å evaluere opp mot min studie. Disse tre er: *begrepsvaliditet*, *intern validitet*, *ekstern validitet*. Etter å ha sett på disse vil jeg også se nærmere på *økologisk validitet* (norsk oversettelse av *ecological validity*) som blir lagt frem hos Bryman (2012, s. 48).

Begrepsvaliditet handler om i hvilken grad den gjennomførte «målingen» samsvarer med de teoretisk definerte begrepene (Kleven & Hjardemaal, 2018, s. 96). Kleven og Hjardemaal (2018, s. 98) forteller at et kritisk spørsmål man kan stille for å evaluere begrepsvaliditet er «Hvordan er begrepene operasjonalisert?». Dette innebærer hvordan en har valgt å måle det en ønsker å måle. Det jeg ønsker å undersøke i min studie er problemstillingen: *Programmering*

som en matematisk representasjon. For å kunne undersøke denne problemstillingen har jeg laget to forskningsspørsmål: (1) *Hvordan kan programkode betraktes som en matematisk representasjon?* og (2) *Hvordan kan programmering støtte opp under elevers matematiske verbale representasjon?* Informantelevene arbeider i undervisningsøkten med matematiske programmeringsoppgaver. Forskningsspørsmålene mine svarer jeg på ved å operasjonalisere (1) i hvilken grad programkoden som elevene produserer kan betraktes som en matematisk representasjon, og (2) hvordan elevenes verbale representasjonsform er påvirket av programmeringsarbeidet. Denne operasjonaliseringen er gjort ved å samle inn data i form av lyd-, skjerm- og videoopptak. Analysen av disse dataene bidrar til å svare på forskningsspørsmålene som igjen bidrar til å belyse problemstillingen.

De empiriske dataene mine vil igjen være basert på hvordan jeg har valgt å operasjonalisere begrepene på. Empirien kan brukes til å suppleres på en rasjonal vurdering av begrepsvaliditeten, da det ikke er mulig å estimere begrepsvaliditet gjennom et tallmessig uttrykk (Kleven & Hjardemaal, 2018, s. 108). Jeg mener at den empirien jeg har fått på bakgrunn av min operasjonalisering danner et godt grunnlag for undersøkelsen av programmering som en matematisk representasjon. Samtidig kan jeg ikke utelukke at det alltid vil finnes noen former for feil når man driver med slike «målinger» som jeg har gjort. Begrepsvaliditeten til min studie kunne derfor vært styrket dersom jeg for eksempel hadde gjennomførte flere ulike operasjonaliseringer. Kleven og Hjardemaal (2018) sier og at det må vurderes om det jeg måler stemmer overens med det jeg ønsker å finne teoretisk. Dette blir vanskelig å vurdere i min setting hvor det ikke er et godt teoretisk grunnlag som ser på programmering som en matematisk representasjon.

Indre validitet sier noe om årsaksforholdet mellom to variabler som inngår i studien (Kleven & Hjardemaal, 2018, s. 117). For å beskrive årsaksforholdet trekker Kleven og Hjardemaal (2018) inn *årsaksrelasjoner*: «En relasjon mellom to variabler kalles en årsaksrelasjon dersom den ene variabelen har en viss påvirkning på den andre» (Kleven & Hjardemaal, 2018, s. 118). For min studie vil denne årsaksrelasjonen være tilknyttet det jeg har observert, og de slutningene jeg har tatt på bakgrunn av disse observasjonene. Kleven og Hjardemaal (2018, s. 132) forteller at en av styrkene med kvalitative studier er at det er bedre grunnlag for å gjøre tolkninger av datamaterialet, som bidrar til å øke den indre validiteten. Samtidig er den indre validiteten avhengig av at jeg som forsker kan reflektere over om mine funn svarer til det jeg har observert, eller om det kan finnes andre alternative tolkninger. Noen av mine funn er basert på observerte enkeltsituasjoner som jeg fant interessante. Det å trekke ut et funn fra én situasjon, kan tyde på

at funnet er usikkert, og at det kan finnes flere alternative tolkninger av den observerte situasjonen. Dette kan bidra til å svekke min studies indre validitet, og gjøre resultatene usikre. Samtidig er det slik at jeg i denne studien undersøker et felt med relativt lite tidligere forskning, og Kleven og Hjardemaal (2018, s. 132) sier at usikker kunnskap kan danne grunnlag for fornuftige konklusjoner. Jeg vil påstå at mine data ga meg et rikt og nyansert bilde av elevkonteksten, noe som gjør at de slutningene jeg har tatt virker sannsynlige. Samtidig vil jeg ikke utelate at det kan finnes alternative tolkninger som jeg ikke har tatt betraktning til i min studie.

Ekstern validitet handler om resultatene fra studien kan generaliseres utenfor den spesifikke forskningskonteksten det har røtter i (Bryman, 2012, s. 47). Hvis min forskning *ikke* er av ekstern validitet, kan konklusjonene mine kun knyttes til den informantklassen jeg har undersøkt, og kan ikke generaliseres til andre klasser eller andre elever. Bryman (2012, s. 69) stiller spørsmål til den eksterne validiteten tilknyttet casestudier. Hvordan kan empiri fra én informantklasse brukes til å danne resultater som kan generaliseres over til for eksempel andre klasser? Kvalitativ forskning kjennetegnes ved at det ikke er generaliserbart, og Bryman (2012, s. 408) bruker dette for å skape en kontrast mellom kvantitativ og kvalitativ forskning. Hvor kvantitative forskeren vil ha resultater som er generaliserbare, vil den kvalitative forskeren fokusere på å forstå hva som skjer med hensyn på konteksten til hvor forskningen finner sted (Bryman, 2012, s. 408). Dette kan også ses på i sammenheng med hvordan man valgte ut informanter. Jeg gjorde et kriteriebasert utvalg, og siden informantene ikke var tilfeldig valgt ut, er det derfor ikke mulig å generalisere til en større populasjon (Gleiss & Sæther, 2021, s. 39). Samtidig påpeker Gleiss og Sæther (2021, s. 207) at kvalitativ forskning innebærer en annen form for generalisering, kalt for analytisk generalisering. En slik generalisering handler om at de systematiseringene som har blitt gjort i analyseprosessen, kan ha relevans i andre settinger enn akkurat der hvor undersøkelsen ble gjennomført. I og med at jeg har utført en kvalitativ forskning basert på data fra ett klasserom, kan det være vanskelig å vurdere den eksterne validiteten. Jeg vil fremdeles påstå at min analytiske prosess, og funnene jeg har gjort, kan ha relevans i for eksempel andre klasserom andre steder i landet.

Økologisk validitet handler om i hvilken grad resultatene er overførbare til virkeligheten og til folks hverdagsliv (Bryman, 2012, s. 48). Lav økologisk validitet kjennetegnes ved at forskningen for eksempel blir gjennomført i unaturlige settinger (Bryman, 2012, s. 48). Min datainnsamling ble gjennomført i en undervisningsøkt hvor også faglærer var tilstedeværende. Jeg påstår derfor at gjennomførelsen av innsamlingen ble gjort under en setting som føltes

naturlig for elevene, og som derfor bidrar til høy økologisk validitet. Samtidig kan jeg ikke utelukke at bruk av lyd-, skjerm- og videoopptak, og min rolle som forsker, kan ha påvirket elevene noe som igjen påvirker den økologiske validiteten.

3.7 Etiske perspektiv

De nasjonale forskningsetiske komiteene (NESH) har utviklet forskningsetiske retningslinjer som skal redegjøre for ulike hensyn og forpliktelser, og utdyper ansvaret til forskere, forskningsinstitusjoner og andre forskningsaktører (NESH, 2021, s. 7). I forskning vil det alltid være viktig å ta hensyn til etiske perspektiver tilknyttet prosjektet. Som forsker innenfor lærerutdanningen sentreres forskningen vanligvis rundt mennesker, noe som gjør det ekstra viktig å ta betraktning til de etiske forholdene. I mitt prosjekt har jeg hatt tilgang til både lyd-, skjerm- og videoopptak av informantelever og faglærer. Derfor forankres mange av mine forskningsetiske forpliktelser i de retningslinjene som blir presentert i del B) hos NESH (2021, s. 18) om *hensyn til personer*. Her sier de innledningsvis at «forskere har ansvar overfor alle personer som inngår i eller deltar i forskning. Forskere skal respektere deres menneskeverd og ta hensyn til deres personlige integritet, sikkerhet og velferd. Forskningsdeltakere skal som hovedregel være informert og samtykke til å delta i forskning» (NESH, 2021, s. 18).

På forhånd av datainnsamlingen ble det gitt en muntlig presentasjon om forskningen jeg skulle gjennomføre. Faglærer og elevene fikk også utdelt et informasjonsskriv med et samtykkeskjema. Dette skjemaet kan sees i vedlegg 2. Siden elevene var over 16 år, kunne de signere samtykkeskjemaet selv. I dette skrivet ble det informert om:

- Formålet med forskningen
- Hvem som står ansvarlig for prosjektet
- Hvorfor eleven har blitt spurt om å delta
- Hva det innebærer for eleven å delta
- At det er frivillig å delta
- Lagring og bruk av personvernsopplysninger
- Elevens rettigheter

Før datainnsamlingen ble gjennomført ble forskningsprosjektet også registrert hos RETTE som er Universitetet i Bergen sitt system for oversikt og kontroll med behandling av personopplysninger i forsknings- og studentprosjekter (UiB, 2023). Både registeringen i RETTE og innsamling av samtykkeskjema fra elevene ble fullført før datainnsamlingen ble gjennomført i november 2022.

Både før under og etter datainnsamlingen måtte jeg også følge NESH (2021, s. 22-24) sine retningslinjer om anonymitet og om konfidensialitet og taushetsplikt. I og med at jeg var tilstedeværende under datainnsamlingen og hadde videoopptak av elevene ble ikke datainnsamlingen gjort anonymt. Min jobb ble derfor å anonymisere. I følge NESH (2021, s. 23) innebærer anonymisering å fjerne forbindelsen mellom personer og informasjon slik at opplysningene ikke kan spores tilbake til individet. For å bevare anonymitet har jeg sørget for å ikke inkludere navn eller bilder av informantelever, faglærer eller skole i presentasjon av datamaterialet. All datamaterialet har også blitt behandlet konfidensielt, og har blitt lagret forsvarlig i et system som krever tofaktorautentisering.

4.0 Funn

I dette kapittelet vil jeg presentere studiens funn. Etter analyseprosessen endte jeg opp med fire funn knyttet til programmering som en matematisk representasjon. Disse funne er henholdsvis: (1) *elever behandler programkode som noe manipulativt*, (2) *elever forstår ikke nødvendigvis symbolbruk i Python*, (3) *elever skriver programkode for seg selv* og (4) *programmeringsobjekter og programmeringsbegreper kan mediere elevenes verbale representasjoner av matematiske objekter*.

Jeg vil presentere hvert av disse funnene i dette kapittelet. Funn (1)-(3) vil bli presentert i egne delkapitler. Funn (4) vil også bli presentert som et eget delkapittel, men vil i tillegg ha tilhørende underkapitler for hvert matematiske objekt. Til alle funn bortsett fra (3) vil jeg inkludere eksempler fra elevenes programkode og tekstutdrag fra elevdialogene, for å underbygge funnet.

Når jeg presenterer tekstutdrag vil jeg gjøre dette gruppevis, hvor jeg viser til dialog fra en gruppe om gangen. Dette blir kun gjort for å systematisere utdragene. I hver gruppe har elevene blitt navnsatt som E1, E2 eller E3 (og E4 i de tilfellene hvor gruppen har flere enn tre elever). Jeg vil gjøre leser oppmerksom på at for eksempel E2 kun er samme person i den gruppen det henvises til. Altså er E2 i gruppe 1 ikke den samme personen som E2 i gruppe 4.

4.1 Elever behandler programkode som noe manipulativt

Mitt første funn er at *elever behandler programkode som noe manipulativt.* Hvor elever kopierte og limte inn deler med programkode for å lage nye programmer. Dette funnet ble gjort under analyseprosessen, hvor jeg observerte noe interessant når gruppe 1 arbeidet med oppgave 4, som handler om å tilnærme et areal ved bruk av trapesmetoden. Som kontekst har gruppen har allerede klart å lage et program som løser oppgave 2 og 3, hvor de skulle tilnærme arealet under grafen med å bruke venstremetoden og høyremetoden.

For å få en bedre forståelse av utdragene jeg skal presentere vil jeg starte med å gi litt bakgrunnsinformasjon. Oppgave 2, 3 og 4 har til felles at de bruker det samme funksjonsuttrykket $f(x) = \frac{1}{2}x^2 + 4$, på det samme intervallet $[0, 2]$. På dette intervallet er funksjonen voksende. Venstremetoden vil derfor alltid gi et tilnærmet areal som er mindre enn det eksakte arealet, og kan derfor omtales som *sum under*. Høyremetoden vil gi rektangler med en større høyde enn venstremetoden, og vil derfor få et tilnærmet areal som alltid er større enn det eksakte arealet. Høyremetoden kan derfor omtales som *sum over*. I venstre- og høyremetoden vil man altså bruke rektangler av samme bredde, men med ulik høyde. Dette kan

kombineres for å finne arealet av et trapes hvor man adderer sammen disse to rektanglene og deler på to.

I eksempelet jeg nå skal presentere kan vi se at eleven E1 utfører manipulasjoner av programkoden fra de forrige oppgavene, for å lage et program som summerer areal av trapes. Utdraget nedenfor viser hvordan E1 forklarer ideen sin.

1. E1: Okey så nå skal vi bruke både under grafen og over grafen.

2. E2: Da kan du jo få nøyaktig kan du ikke det? Hvis du har en lineær graf.

3. E1: Jo. Da trenger du vel bare å kopiere den der, også lime den inn. Også må vi ha to for in range og to h'er. Fordi at da får vi jo to *utydelig*

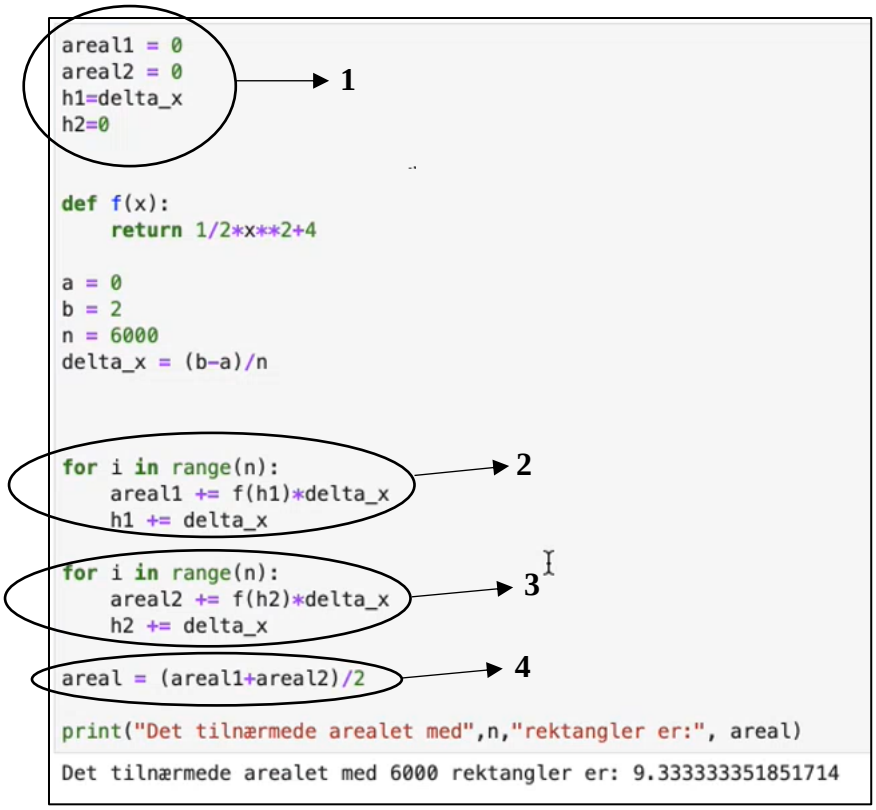
4. E2: Skal du ha to der?

5. E1: Nei hvis du går litt ned, også kopier den der, nei hele den, også med h'en, også kopier den (kopierer følgende kode:

```
h=delta_x

for i in range(n):
    areal += f(h)*delta_x
    h += delta_x
```

), også lim den inn under. Fordi at der har vi jo for over grafen, også må vi ha en for under grafen, også må vi plusse de to sammen og dele de på to.

I utdraget over gir E1 instrukser til E2 om hvordan programkoden skal manipuleres. Vi ser først i linje 1 at E1 har en idé om å bruke «under grafen» og «over grafen», som henviser til venstre- og høyremetoden. Videre til linje 5 uttrykker E1 hvordan programkoden skal manipuleres. Her legger jeg spesielt merke til ordene «kopier den», «lim den inn», «pluss de to sammen» og «del de på to». Når E1 sier «den» så henvises det til en bit med programkode som de skrev i oppgaven om høyremetoden. Denne biten av programkoden blir kopiert og limt inn i et nytt programmeringsvindu, for å brukes i programmet som løser oppgave 4. Programkoden elevene ender opp med til slutt er presentert i figur 10 nedenfor.



Figur 10. Programkode til gruppe 1 som arbeider med trapesmetoden.

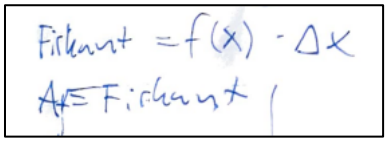
I denne programkoden har kodebiter fra de tidligere oppgavene blitt satt sammen for å svare på den nye oppgaven. For å enklere snakke om hva som skjer i programkoden har jeg satt sirkel rundt ulike linjer med kode. Jeg vil starte med å se på sirkel 2 og 3. Måten E1 uttrykker seg på i elevdialogen, gir meg inntrykket av at programkoden representerer fysiske rektangler for E1. Den første for-løkken i sirkel 2, representerer rektangler fra høyremetoden (sum over), og den andre for-løkken i sirkel 3, representerer rektangler fra venstremetoden (sum under). Sirkel 1 viser til de ulike variablene for de ulike metodene. I sirkel 4 blir arealet til de to rektanglene fra sirkel 2 og 3, addert sammen og delt på to for å gi areal av trapes på denne formen:

Biter av programkode blir altså manipulert ved at det *kopieres* og *limes inn* i et og samme program hvor de adderes og deles for å lage trapes som tilnærmer arealet under grafen.

4.2 Forståelse av symboler i Python

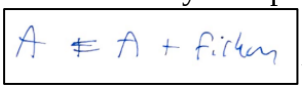
Mitt andre funn er at *elever ikke nødvendigvis forstår symbolbruk i Python*. Her er det symbolene «=» og «+=» jeg tar utgangspunkt i. Dette funnet ble gjort når jeg analyserte gruppe 5, hvor det oppsto tre situasjoner som underbygger dette funnet. Denne gruppen besto av tre elever hvor E2 hadde mer programmeringserfaring enn E1 og E3. I elevdialogen stiller E1 og E3 spørsmål til enkelte deler av programkoden som skrives av E2. Jeg vil nå legge frem de tre situasjonene.

Den første situasjonen er når gruppen arbeider med papiroppgaven, som går ut på at de skal planlegge programmet de skal lage. E2 skriver ned følgende på papiret sitt:



Deretter stiller E3 et spørsmål, som fører til elevdialogen vist nedenfor:

1. E3: Hva betyr a pluss er lik?

2. E2: Det betyr at a pluss, eh ja. Det er det samme som a er lik a pluss firkant (E2 skriver 

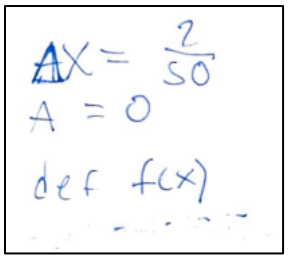
). Så vi har liksom den forrige a verdien, så hvis vi først får for eksempel 1 som a, også blir firkant, eller ja, nei. Først får vi a som er null, så får vi en firkant som har areal på 1, for eksempel. Da har vi a som er null, pluss 1, også gjør vi det igjen, så får vi 1 pluss 1, og da er jo a 1.

3. E3: Ja, men jo det ga litt mening.

4. E2: Det er i hvert fall det samme som det, bare lettere skrevet.

Det første utdraget ovenfor viser at E2 har skrevet ned symbolene «$A+= firkant$» når han skisserer en idé til hvordan programmet skal se ut. I linje 1 i dialogen spør E3 hva disse symbolene betyr. I linje 2 starter E2 med å forklare at det er det samme som å skrive «$A = A + firkant$».

Den andre situasjonen oppstår også når gruppen arbeider med papiroppgaven. E2 har skrevet følgende på papir:



Deretter stiller E1 et spørsmål, som fører til elevdialogen vist nedenfor:

1. E1: Men a er lik null, hvorfor er det sånn?

2. E2: Fordi arealet på begynnelsen da har vi ikke gjort dette her, så hvis vi skal finne første firkanten. Eller først har vi arealet null sant, fordi vi ikke har funnet noen arealer av de firkantene. Mens når vi begynner å finne den første firkanten, så legger vi til den da. Jeg er ikke så god til å forklare det. Men vi begynner med a er lik null fordi vi ikke har funnet noen arealer, så bare legger vi til et og et areal av rektanglene.

På papiret har E2 skrevet $A = 0$. I linje 1 i dialogen spør E1 hvorfor det er slik. I linje 2 fra utdraget over ser vi at E2 forklarer hvorfor han har skrevet $A = 0$. For å forklare hvorfor hen har skrevet det slik beskriver hen en prosess med å si «først har vi», etterfulgt av «så legger vi til». Det å skrive at $A = 0$, eller si at areal er lik null, kan fra et algebraisk perspektiv virke rart. I programmeringsverden er dette derimot en helt naturlig ting å gjøre for å definere en variabel å kunne oppdatere variabelens verdi underveis i som programmet kjører.

I den tredje sitasjonen arbeider gruppen med å programmere på PC. De har skrevet ferdig en programkode, hvor de har startet med å si at variabelen $areal$ er lik null, «$areal = 0$». Etter hvert stiller E1 et spørsmål om hvorfor arealet ikke er noe annet. Dialogen er presentert i utdraget nedenfor:

Gruppen har skrevet følgende programkode, legg merke til «areal = 0» øverst i koden:

```
a = float(input("Hva er a"))
b = float(input("Hva er b"))
n = 0
delta_x = (b-a)/n
x = 0
areal = 0

def f(x):
    return 1/2*x**2+4
for i in range (0, n):
    rektangel = f(x)*delta_x
    areal += rektangel
    x += delta_x

print("Det tilnærmede arealet med",n,"rektangler er:", areal)
```

1. E1: Må ikke arealet være *utydelig*

2. E2: Nei, men arealet, du må jo begynne på null på arealet, også må vi..

3. E3: Det er liksom det som er problemet med fra matte som jeg skriver på papiret mitt. Der ville jeg jo bare skrevet areal er lik ikke sant og hatt det som en variabel og bare gjort matten selv.

4. E2: Men okey, areal begynner som null fordi først finner vi arealet til det første rektangelet, så legger vi til det på areal, også finner vi areal på neste også legger vi til det på areal, og det gjør vi helt til programmet er ferdig.

5. E3: Ja, det gir for så vidt mening, men det er bare det som er liksom omstilling av vanlig matte, så man må, ja.

Dette utdraget starter i linje 1 med at E1 lurer på hvorfor ikke skal være noe annet. Akkurat *hva* annet er usikkert på grunn av utydelig dialog, men det virker rimelig å anta at det er noe annet enn null. E2 starter med å forklare før han blir avbrutt av E3 som i linje 3 uttrykker et problem med programmering vs. «matte som jeg skriver på papiret mitt». Det er ikke sikkert hva E3 mener med «matte på papiret», men jeg påstår det er rimelig å anta at dette henviser til algebraiske notasjoner som elevene er vant til å bruke ellers i matematikken. Dette gir også mening med tanke på hva E3 sier i linje 5 om «omstilling av vanlig matte».

Disse tre situasjonene fra elevgruppen underbygger funnet om at elever ikke nødvendigvis forstår symbolbruk i Python. Dette viser seg i hvordan E1 og E3 lurer på hva «+=» betyr, og hvorfor man starter med å skrive «$areal = 0$», og at dette er en omstilling fra vanlig matematikk.

4.3 Elever skriver programkode for seg selv

Det tredje funnet er at elever skriver programkode for seg selv. Med dette mener jeg at selv om elever satt i grupper for å samarbeide om programmeringsoppgaven, så var det tilfeller hvor kun én på gruppen skrev programkode uten å kommunisere med de andre på gruppen, eller at

elever på gruppen tok frem sin egen PC for å teste sin egen kode. Funnet ble gjort på bakgrunn av observasjoner som ble gjort både under og etter gjennomføringen av datainnsamlingen.

De første observasjonene ble gjort under gjennomføringen av datainnsamlingen. Til tross for at klassen hadde fått beskjed om å kun ha én PC per gruppe, la jeg merke til at elever på enkelte grupper tok frem sin egen PC. Når jeg spurte hvorfor de hadde tatt frem sin egen PC, svarte de at det enten var fordi de skulle prøve å skrive sin egen programkode, eller sjekke om resultatene deres var korrekt ved å bruke GeoGebra. Dette ble lagt spesielt godt merke til hos gruppe 7 hvor plutselig alle elevene på gruppen jobbet på sin egen PC.

I transkriberingsfasen ble det også gjort observasjoner som jeg koblet opp mot det jeg observerte under selve gjennomføringen. Gjennom opptakene kunne jeg observere at det hos de fleste gruppene var et tidspunkt hvor én elev skrev programkode alene. Dette observerer jeg først og fremst med at det er liten eller ingen verbal aktivitet i gruppen. Samtidig ser jeg at det er aktivitet på skjermopptaket, og gjerne bare én person å se via webkameraet.

Disse observasjonene underbygger funnet om at elever skriver programkode for seg selv. Grunnen til at trekker dette med som et funn er fordi det var noe som var gjentakende på de fleste gruppene. For meg virker det som en interessant handling knyttet opp mot problemstillingen: Programmering som en matematisk representasjon.

4.4 Programmeringsobjekter og programmeringsbegreper kan mediere elevenes verbale representasjoner av matematiske objekter

Det fjerde funnet er at programmeringsobjekter og programmeringsbegreper kan mediere elevenes verbale representasjoner av matematiske objekter. Dette funnet ble gjort når jeg i del 2 og 3 av analyseprosessen undersøkte hvordan matematiske objekter ble uttrykt i elevenes verbale representasjon. Begrepet «å mediere» betyr i denne sammenhengen at bruk av programmeringsobjekter og programmeringsbegreper har en innvirkning på elevenes dialog omkring de matematiske objektene.

For å underbygge dette funnet vil jeg strukturere dette delkapittelet i underkapitler som tar for seg hvert av de matematiske objektene. Disse matematiske objektene ble identifisert i analysekapittelet og var (1) funksjon, (2) bredde, (3) areal av rektangel, (4) sum av areal og (5) x-verdier. Alle disse matematiske objektene ble funnet ved å undersøke den visuelle og symbolske representasjonen av numerisk integrasjon.

I hvert av disse underkapitlene vil jeg legge ved utdrag fra elevdialog.

4.4.1 Funksjon

Funksjoner i programmering har mange likhetstrekk med hvordan man er vant til å jobbe med funksjoner enten algebraisk på papir eller i GeoGebra på PC-en. Funksjonen tar inn en verdi, og gir ut en annen verdi. I Python kan man ikke skrive inn funksjonen slik som man er vant til med den symbolske måten, altså som for eksempel $f(x) = \frac{1}{2}x^2 + 4$. I Python må man *definere* en funksjon og si hvilke verdier den skal ta inn, og hva som skal gjøres med denne verdien, før den til slutt *returnerer* den ferdigbehandlede verdien. Dette kan se slik ut:

```
1   def f(x):
2       return (1/2)*x**2+4
```

I linje 1 sier vi til programmet at vi har en funksjon som har navnet «f», som tar inn en verdi «x». Programmet skal deretter ta verdien til «x», opphøye den i andre ($x ** 2$), dele den på 2 og addere på 4 før den «nye» x-verdien blir returnert.

Det å definere funksjoner i Python på denne måten er vanligvis noe man lærer ganske tidlig i programmeringsfasen. Noen grupper hadde mer kontroll enn andre, og derfor var det ikke muntlig diskusjon rundt dette matematiske objektet i alle gruppene. De følgende gruppene diskuterte funksjonen.

Gruppe 2:

I dette utdraget kan vi se hvordan E2 forklarer til E1 hvordan man lager en funksjon i Python:

1. E2: Også må du også definere funksjonen.

Elev 1 starter å definere funksjonen:

```
def f(x)=
```

2. E1: def f av x er lik, også må vi..
3. E2: Du må skrive kolon, istedenfor er lik.
4. E1: Må jeg det?
5. E2: Ja, også må du trykke enter.
6. E1: Jaja.
7. E2: Også return, også må vi skrive funksjonen.
8. E1: Ja, det må jeg jo gjøre!

E1 prøver å kopiere inn funksjonsuttrykket som er gitt i oppgaveteksten:

Oppgave 2

Vi ser på følgende funksjon: $f(x) = \frac{1}{2}x^2 + 4$

Men når den kopieres inn i programmeringsruten så blir det seende slik ut:

```
def f(x):
    return 12x2+4
```

9. E2: *sier noe utydelig til elev 1*
10. E1: Ja det var det ja. Vent litt jeg må vel sikkert bruke.. kan jeg ikke bare gjøre slik som dette her, multiplisert med x opphøyd i andre pluss, ja pluss 4.

E1 skriver inn følgende:

```
def f(x):
    return 0.5*x**2+4
```

.

11. E1: Sånn. Okey hvis vi runner nå så blir det sikkert masse feil (trykker på «run»). Nei, det ble det ikke, glem det!

I utdraget fra gruppe 2 ser vi at E1 ikke vet helt hvordan man skriver inn funksjoner i Python. E2 har litt mer kontroll og kommer med noen råd til hvordan koden skal skives. Programmeringsobjekter og programmeringsbegreper kommer til uttrykk her. I linje 3 påpeker E2 en syntaks feil, hvor er lik «=» må byttes ut med kolon «:». Videre i linje 5 og 7 så forteller E2 om hvordan resten av funksjonen skal skrives «også må du trykke enter», «også return, også funksjonen». Dette kan knyttes til programmeringsobjektet som handler om *rekkefølger*. Programmet ville ikke lest koden som en funksjon hvis ikke rekkefølgen hadde vært slik som E2 beskriver her. I dialogen brukes også uttrykk som «definere» i linje 1, «return» i linje 7, og «runner» i linje 11. Dette er programmeringsbegreper som medierer elevdialogen når de snakker om det matematiske objektet funksjon i arbeid med programmering.

Gruppe 4:

I dette utdraget snakker elevene om hvor funksjonen skal plasseres i programmet:

1. E3: Jo, ja det gjør vi. Okey vent litt da, skal jeg skrive sånn definisjon, eller den der f av x her nede eller skal jeg skrive den her oppe?
2. E2: Kan du ikke bare definere funksjonen først kanskje?
3. E1: Ja.
4. E3: En halv, det må jeg kanskje ta i parentes, må jeg ikke det?
5. E1: Jo jeg tror det.

I utdraget ser vi i linje 1 hvordan E3 lurer på hvor hen skal definere funksjonen. E2 sier i neste linje at det kan blir gjort først. Her er det programmeringsobjektet *rekkefølger* som virker inn på de verbale representasjonene. En annen ting å bemerke seg fra dette utdraget er hvordan E3 i linje 1 sier «skal jeg skrive sånn definisjon». Her henviser eleven til operatoren *def*, som brukes når en funksjon skal defineres i Python. Men eleven sier ikke direkte «def», men uttrykker heller «sånn definisjon». «Definisjon» blir i denne konteksten et programmeringsbegrep. Det samme ser vi i linje 2 hvor E2 sier «kan du ikke bare definere funksjonen».

4.4.2 Bredde

Det andre matematiske objektet som er sentralt i arbeid med numerisk integrasjon er det som jeg har valgt å kalle for «bredde». Her kunne det også blitt brukt andre begreper. Fra et algebraisk perspektiv bruker man gjerne symbolet Δx for å representere bredde. Symbolet Δx leses som «delta x», og det er som oftest slik elevene omtaler det i elevdialogene. Måten man beskriver bredden, eller delta x, i programmering kan ha en del likheter med hvordan man ville gjort det algebraisk. Derfor er de verbale representasjonene rundt dette matematiske objektet ikke spesielt avhengige av programmeringsspråk og programmeringsobjekter. Jeg henviser her kun til én gruppedialog.

Gruppe 2:

I dette utdraget diskuterer elevene hvordan de skal uttrykke variabelen «delta_x» til oppgave 1a):

1. E2: Oja her skriver vi intervallene. Så da skriver vi 0 på a, og 2 på b.

2. E4: b er 4

3. E1: Så er n lik 10, ifølge det som står her.

4. E2: Og delta_x er..

5. E1: En halv.

6. E2: Eh, nei men det, oja, nei men det er på figuren over. Delta x er jo 4 delt på n. Fordi det er jo et intervallslutt delt på antall rektangler.

7. E1: Oja, ja ja selvfølgelig. Skal jeg bare skrive 4 delt på 10?

8. E4: Nei bare ta b delt på n.

9. E2: Ja fordi da blir den mer global liksom.

10. E1: Ja. Sånn, hvis vi bare kjører den da.

11. E2: Da får vi 0.4, og det er jo sant.

De har da skrevet følgende programkode til oppgave 1a):

```
a = 0
b = 4
n = 10
delta_x = b/n
print("delta_x =", delta_x)
```

```
delta_x = 0.4
```

I utdraget over arbeider elevene i gruppe 2 med å finne et uttrykk for delta_x for oppgave 1a), hvor de får oppgitt at $a = 0, b = 4$ og $n = 10$. I linje 1-3 starter gruppen med å tilegne verdier til variablene a, b og n. Når det skal gis en verdi til variabelen delta_x foreslår E1 i linje 7 å skrive inn «4 delt på 10». E4 mener derimot i linje 8 at «bare ta b delt på n», og E2 støtter dette i linje 9 fordi med å si at den blir «mer global». Det er disse to linjene som trekker inn programmeringsaspekter i dette utdraget. I og med at delta_x avhenger av de andre variablene, kan den likeså godt bare uttrykkes ved disse variablene. Skulle en av disse variablene få en ny verdi, så vil delta_x automatisk få en ny verdi. E2 bruker begrepet «global», som inngår i programmeringsobjektet *operator*. Dette en operator i Python som gjør at en variabel blir lettere tilgjengelig større deler av programmet. I akkurat dette programmet så har ikke elevene behov for å bruke denne operatoren, men programmeringsobjektet medierte fremdeles den verbale representasjonen av det matematiske objektet bredde. Det kan også rettes en liten kommentar til at uttrykket for delta_x i programkoden deres er feil, det skulle vært $(b - a)/n$, men siden $b = 0$ blir svaret fremdeles gir programmet fremdeles ut korrekt svar.

4.4.3 Areal av rektangel

Numerisk integrasjon handler om å summere sammen arealet til mange rektangler. Da blir det å kunne finne arealet av et rektangel en naturlig del av prosessen.

Gruppe 5:

Dette utdraget starter med at elevene diskuterer hvordan de skal finne arealet av det første rektangelet:

1. E2: Hva hadde vi gjort for å finne det første rektangelet?

2. E3: Da hadde vi tatt f av null ganger med null. x ganger f av x egentlig.

3. E2: Men er det ganger null eller er det ikke ganger delta_x da?

4. E3: Jo, delta_x mener jeg.

5. E1: Ja for delta_x er på en måte grunnlinjen til det.

6. E2: Også for å finne neste så hadde vi hatt f av 1 ganger delta_x.

7. E3: Først f av null ganger delta_x, så blir det delta_x ganger f av delta_x også ville det vært delta_x ganger f av to delta_x, blir det ikke sånn?

I dette utdraget ser vi i linje 1 at E2 spør om hvordan de skal gå frem for å finne det første rektangelet, altså arealet av ett rektangel. Videre diskuterer de hvordan de skal uttrykke dette, fra linje 2 til 5 blir de enig om at «f av null ganger delta_x» vil være arealet av det første rektangelet. I linje 6 spør E2 hvordan de da skal finne neste areal. Her foreslår E2 «f av 1 ganger delta_x», mens i linje 7 mener E3 at det vil være «delta_x ganger f av delta_x». Her mener jeg at programmeringsobjektet *løkker* har en innvirkning på det som blir sagt i de to siste linjene. Hvor E2 foreslår at neste areal vil være $f(1) \cdot \Delta x$, hvor jeg antar at dette «1» tallet ble brakt inn fordi E2 har tenkt å bruke en «for-løkke» for å finne areal av flere rektangler. En for-løkke går gjennom heltallene 0, 1, 2, …, n-1. Og siden de ble enig om at første areal skulle være $f(0) \cdot \Delta x$, kan det derfor for E2 gi mening at neste vil være $f(1) \cdot \Delta x$. Det blir derimot ikke nevnt noen programmeringsbegreper i dette utdraget.

Gruppe 4:

I dette utdraget diskuterer elevene hvordan de skal finne arealet:

1. E2: Vi har jo definert funksjonen, så hvis vi skriver f av null, vil ikke den i teorien regne det ut?

2. E3: Skal jeg skrive f av null? Men arealet er jo ikke f av null.

3. E2: Areal er lik f av null ganger delta_x.

I utdraget over stiller E2 i linje 1 spørsmål om hvordan en funksjon vil fungere i Python. Eleven spør om ikke «den» vil regne ut $f(0)$, siden funksjonen allerede er definert. Videre bruker eleven dette for å gi et uttrykk for arealet. Her vil jeg si at programmeringsobjektet *datastrukturer*, spiller inn på E2 sin prosess fra linje 1 til linje 3. Eleven måtte først vite at «f av null» ville gi en verdi, før det igjen kunne brukes for å uttrykke arealet av et rektangel.

4.4.4 Sum av areal

Det er selve summeringsprosessen som gjør at programmering fungerer utmerket i numeriske metoder. Dette er det matematiske objektet som muligens skaper størst forskjell mellom programmering og algebraiske uttrykk. Til dette matematiske objektet vil jeg presentere utdrag fra gruppe 3 og gruppe 5. Disse gruppene hadde to ulike metoder for å bergene summen av arealene til rektanglene. Gruppe 3 valgte å legge til alle arealene i en liste, og summere sammen denne listen i slutten av programmet. Gruppe 5 valgte å summere arealene inn i en variabel fortløpende, for å så skrive ut denne variabelen i slutten av programmet. For å gi mer kontekst kommer jeg også til å presentere programkoden gruppene endte opp med etter å ha arbeidet med oppgave 2, hvor de skulle tilnærme areal ved hjelp av venstremetoden.

Gruppe 3:

De verbale representasjonene til gruppe 3 for dette matematiske objektet vil bli delt opp i to tekstbokser. Dette er fordi de verbale representasjonene blir uttrykt ved ulike deler i programmeringsprosessen. Jeg vil starte med å gi et utdrag knyttet til papiroppgaven før jeg gir et utdrag fra når de arbeider på PC.

I utdraget under arbeider gruppen med papiroppgavene, hvor de skulle planlegge programmet de skulle lage:

1. E2: Men vi burde egentlig laget det sånn at når den har kjørt løkken en gang, så tar den vare på den verdien vi har fått sant, da tar den vare på det ene rektangelet, også neste gang så blir det et nytt rektangel, og da tar den var på det arealet vi har fått for det, også til slutt tar den summen av alle de rektanglene vi har regnet ut.

…

2. E3: Kan vi ikke si sånn Sn pluss Sn1 pluss Sn2.

3. E2: Ja, men da må vi på en måte lage en ny verdi hver gang da, Sn1 det blir jo en egen verdi og Sn2 det blir jo en egen verdi.

4. E3: Ja, men går det ikke an å få programmet til å printe den 50 ganger så bare legger den sammen for oss?

5. E2: Ja, men da må vi lage en liste, også summere listen, må ikke vi? Tror du ikke det kan gå?

6. E1: Jo.

7. E2: Men da må vi lage en liste, definere en liste.

…

8. E2: Også må vi ta append, L append, og A, er det ikke det, stor a?

9. E1: Skal vi se, L append A med sånn der..

10. E2: Eh ja, sånn kolon, nei sånn parentes er det ikke det?

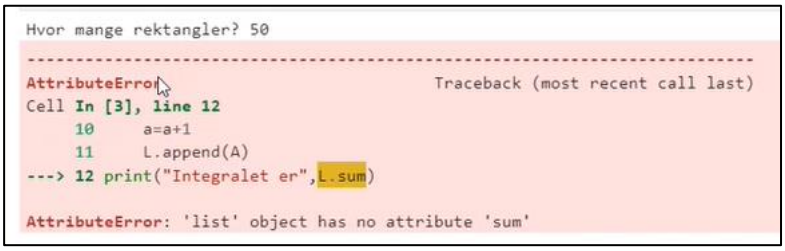
11. E1: Jeg vet ikke om det er parentes eller…

12. E2: Ja men det finner vi ut av, jeg føler ikke det er det det står på. Også må vi bare ha summen av hele listen, jeg vet ikke helt hvordan man tar sum av hele listen, er det ikke L sum?

Utdraget over får frem ulike deler av planleggingsprosessen til gruppen. I linje 1 starter E2 med å presentere en idé til hvordan oppgaven kan løses. I linje 5 trekker E2 inn programmeringsbegrepet «liste». «Liste» er også en måte å behandle verdier på og inngår derfor i programmeringsobjektet *datastrukturer*. Når man arbeider med lister finnes det også ulike operatorer som kan være nyttige å bruke. To av disse operatorene kommer til uttrykk i linje 8, og linje 12. Dette er operatorene *append* og *sum*, hvor append legger til en verdi i listen, og sum summerer sammen verdiene i listen.

Etter at gruppen var fornøyd med å planlegge programkoden prøvde de å skrive programmet på PC-en. Utdraget under viser gruppens dialog etter å ha fått opp en feilmelding.

Gruppen prøver å kjøre et program, men de får følgende feilmelding:



13. E1: L sum

14. E2: L.sum er ikke en greie. Har du noe hvordan vi kan summere..

15. E1: Vi bare definerer L sum da, kan vi ikke det?

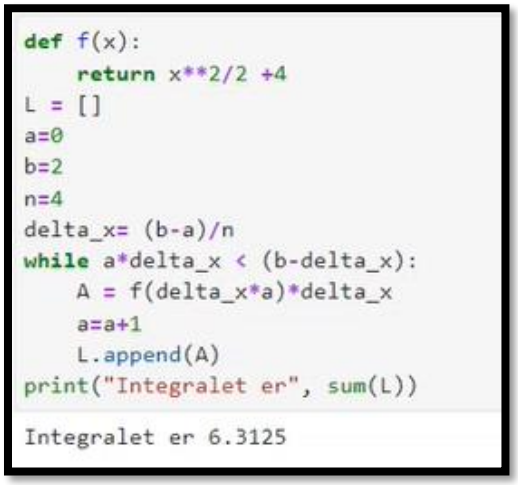
16. E2: Nei, men hvordan summerer vi en liste? (de spør lærer om hjelp)

17. E1: Det er L sum sant?

18. L: Ta sum også parentes L istedenfor.

Elevene bytter og får et program som fungerer.

I dette utdraget ser vi at gruppen får en feilmelding når de kjører programmet sitt. Det viste seg at problemet var knyttet til *sum*-operatoren som var brukt feil. Selv om ideen er korrekt, så må den skrives på en eksakt måte for at programmet skal kunne tolke det. Etter å ha fått hjelp av lærer i linje 18 fikk gruppen produsert programkoden som vist under.



```
def f(x):
    return x**2/2 +4
L = []
a=0
b=2
n=4
delta_x= (b-a)/n
while a*delta_x < (b-delta_x):
    A = f(delta_x*a)*delta_x
    a=a+1
    L.append(A)
print("Integralet er", sum(L))
```

```
Integralet er 6.3125
```

Programkoden vist over er resultatet av gruppe 3 sitt arbeid med oppgave 2, hvor de skulle tilnærme arealet ved bruk av venstremetoden. I en av de øverste linjene kan vi se at de har skrevet «$L = [\,]$», dette er listen hvor de legger til arealene til alle rektanglene. Programkoden gir en feil tilnærming for arealet med fire rektangler, men det er grunnet en feil i while-løkken, som jeg ikke vil ta i detaljer.

Gruppe 5:

I utdraget under arbeider gruppen med papiroppgaven:

1. E2: Okey. Nei, men vi skal jo til sammen ha 50 rektangler.. da er det jo for, ehh, ja. Range da. Det er litt vanskelig å skrive på papir. Ja ehm. Vi må ha en eller annen form for løkke, også må vi plusse på delta x hver gang, hvis det gir mening. Jeg vet ikke helt hvordan jeg skal forklare det.
2. E3: Ja det gir mening at du liksom begynner på f av null, også går det neste til f av null pluss delta_x, også er det f av null pluss delta_x pluss delta_x, også er det altså ja. Så det blir jo, altså f av null pluss, ehm, n minus 1 gange delta_x. Eller noe sånn.

…

3. E3: Hva betyr a pluss er lik?
4. E2: Det betyr at a pluss, eh ja. Det er det samme som $A = A + firkant$. Så vi har liksom den forrige a verdien, så hvis vi først får for eksempel 1 som a, også blir firkant, eller ja, nei først får vi a som er 0. Så får vi en firkant som har areal på 1 for eksempel. Da har vi a som er null, pluss 1, også gjør vi det igjen, så får vi 1 pluss 1, og da er jo a 1.
5. E3: Ja, men jo det ga litt mening.

I utdraget uttrykker E2 i linje 1 at de må ha «en eller annen form for løkke». Her blir ordet «løkke» sett på som et programmeringsbegrep. E3 prøver i linje 2 å forklare denne hvordan denne løkken skulle fungert, hvor det skal plusses på 1 ekstra delta_x, eller «bredde», inne i funksjonsuttrykket for hver gang. Her er det programmeringsobjektet *løkker* som medierer dialogen knyttet til summering av areal. E2 kommer senere med et forslag til kode som er å bruke $A+= firkant$. Dette utdraget ble også tatt frem i funn 4.2 om symboler i Python. Her vil jeg vise til at en variabel kan oppdateres kontinuerlig i programmering, hvor variabelen A vil bli tildelt nye verdier underveis. Dette kan knyttes til programmeringsobjektet *datastrukturer* og *operatorer*.

I det neste utdraget har gruppen startet å arbeide på PC, og E2 har klart å skrive en programkode som fungerer:

6. E2: Nå ble det riktig.
7. E1: Hva gjorde du?
8. E2: Jeg endret at istedenfor å legge til x pluss 1 hele tiden så blir det x pluss delta_x.
9. E3: Okey, så det som skjer nå er at når x er mellom 0 og 50, så vil et rektangel være f av x ganger delta_x?
10. E2: Neinei, Det som skjer er at, okey, for i in range da begynner vi med null, den første verdien er null. Så kjører den gjennom dette her med i er lik null, egentlig endrer ikke det noe særlig. For det der gjør bare at den gjentar det 50 ganger. Ehm, så da blir rektangel, ehm, ja da har vi f av x, og x er null til å begynne med, ganger delta_x, så da blir det basically 2 (fordi $f(0) = 2$) ganger delta_x. Så da legger vi til det arealet til den første firkanten til A, også har vi x pluss er lik delta_x ($x+= delta\_x$), for da får vi at den neste x-verdien..
11. E3: Er den forrige pluss delta_x?
12. E2: Ja. Også når vi er ferdig med 50 gjennomganger på en måte..
13. E3: Da printer den. Så vi har basically lagd et program da.

I utdraget over forklarer E2 i linje 8 og 10 hvordan han har fått programmet til å fungere. I linje 10 forklarer E2 en helt prosess på hvordan programmet fungerer. Eleven bruker blant annet programmeringsbegreper når har sier at «for i in range da begynner vi på null». Programmeringsobjektet som medierer dialogen, er *løkker*. Gruppen endte opp med følgende programkode på oppgave 2, hvor de tilnærmet arealet ved bruk av venstremetoden:

```
a = float(input("Hva er a"))
b = float(input("Hva er b"))
n = int(input("Hva er n"))
delta_x = (b-a)/n
x = 0
areal = 0

def f(x):
    return 1/2*x**2+4

for i in range (0, n):
    rektangel = f(x)*delta_x
    areal += rektangel
    x += delta_x

print("Det tilnærmede arealet med",n,"rektangler er:", areal)


Hva er a 0
Hva er b 2
Hva er n 4
Det tilnærmede arealet med 4 rektangler er: 8.875
```

I programkoden over ser vi at gruppen har startet med å skrive «$areal = 0$». De bruker ikke en liste slik som gruppe 3. Og istedenfor å bruke operatoren «sum()», som gruppe 3 gjorde, har gruppe 5 her summert sammen med å bruke operatoren «+=».

4.4.5 x-verdier

For å beregne arealene av de ulike rektanglene må man endre på hvilken x-verdi man setter inn i funksjonsuttrykket for å få høyden til rektanglene. Hvordan man finner de ulike x-verdiene kan gjøres på forskjellige måter i Python. I en tidligere undervisningstime hadde faglærer vist elevene hvordan de kunne plotte grafer i Python ved hjelp av å importere modulen «Numpy». I denne modulen finnes det to operatorer som kan benyttes for å dele opp et intervall mellom to verdier for å så lagre det som en liste. Disse operatorene heter «arange» og «linspace», og når elevene bruker den i programkoden sin skriver de det som «np.arange()» eller «np.linspace()», hvor «np.» henviser til at de henter operatoren fra modulen Numpy.

Gruppe 1:

I utdraget under stiller eleven E2 et spørsmål til programkode som E1 har skrevet, og E1 forklarer ideen sin:

E1 skriver følgende kode:

```
h=0

for i in range(n):
    areal += f(i+h)*delta_x
    h+=delta_x
```

1. E1: Vil ikke det gå?

2. E2: Men hva gjør h'en?

3. E1: h'en den bare, sånn at vi får en ting, for at den skal jo gå opp med 0.5 hver gang sant, f av x skal jo gå opp med 0.5 hver gang. Men vi kan ikke definere delta_x som at den skal gå opp med 0.5 hver gang, for den skal jo være konstant. Det vil jo bli, det vil jo gå opp med 0.5 hver gang sant, så da må vi definere det som skal stå inni grafen som, så går opp med 0.5 hver eneste gang.

4. E2: Jeg tror det går altså.

5. E3: Det var ikke så dumt.

I utdraget over har E1 funnet en løsning på hvordan de kan få ulike x-verdier som vil bidra til å gi ulike høyder på rektanglene. E1 løser dette problemet med å definere en ny variabel *h* som øker med verdien til variabelen *delta_x* for hver gang løkken kjøres. Hen forklarer tankegangen sin i linje 3. Her er det programmeringsobjektet *løkker* som medierer dialogen. Selv om programkoden til gruppen i dette øyeblikket ikke har et korrekt uttrykk for areal, så vil fremdeles fremgangsmåten for å finne x-verdier være riktig.

Gruppe 2:

I utdraget under jobber E2 med å finne de korrekte x-verdiene for å løse oppgaven:

1. E2: Linspace funker ikke. Oja vent, det er på grunn av..

2. E1: *sier noe utydelig*

3. E2: Nei! Koden går fra topp til bunn, og da gir, x-verdier blir ikke endret på. Jeg tror kanskje vi burde bruke arange.

```
import numpy as np

a = 0
b = 2
n = 4
x_verdier = np.arange(a,b,n)
delta_x  = b/n
```

4. E2: Nå blir den null. Det er på grunn av at dette skal være delta_x fordi dette er intervallet vårt, men da må den under.

(Her bytter E2 ut «n» med «delta_x» inne i arange operatoren, og flytter variabelen «x_verdier» *under* variabelen «delta_x».)

```
a = 0
b = 2
n = 4
delta_x  = b/n
x_verdier = np.arange(a,b,delta_x)
```

5. E2: Ok, så nå skriver den ut riktig.

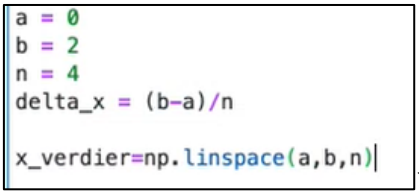
I dette utdraget er det også x-verdier som er det matematiske objektet som er i fokus. Dialogen består for det meste av E2 som snakker til resten av gruppen. Vi ser at i linje 1 og 3 så snakker E2 om «linspace» og «arange» som går inn under programmeringsobjektet *operatorer.* I linje 3 og 4 ser vi også hvordan E2 forteller om plasseringer i form av «topp til bunn» og «da må den under» som viser til programmeringsobjektet *rekkefølger*.

Gruppe 4:

Gruppe 4 valgte å finne x-verdier ved å bruke operatoren *linspace* fra «Numpy»-biblioteket:

1. E2: Men skulle vi kalle den x_verdier da?

2. E1: Ja.

(De skriver inn følgende )

```
a = 0
b = 2
n = 4
delta_x = (b-a)/n

x_verdier=np.linspace(a,b,n)
```

…

3. E2: Vil ikke vi trenge.. for hvis vi lager en liste med x-verdier.

4. E3: Mhm.

5. E2: Vi vil jo ha den i løkken og gange med deltax, var det ikke det?

…

6. E1: Nei fordi se, av denne her så får jo vi da, ehm, mellomrom, gjør ikke vi det da? Jo vi får en liste med alle x-verdiene.

7. E3: Ja.

8. E1: Og de x-verdiene skal vi putte inn i f av, ehh, x, slik at vi får høyden. Også skal vi gange med deltax.

Utdraget over viser til tre ulike tidspunkter hos gruppe 4 hvor de snakker om det matematiske objektet x-verdier (de ulike tidspunktene skilles med «…»). I linje 1 starter E2 med å spørre om de skal kalle den for «x_verdier». Dette kan knyttes opp til hvordan man velger å navnsette variabler i Python, og kan derfor kobles til programmeringsobjektet *datastrukturer*. I linje 3 og 6 bruker E2 og E1 begrepet «liste» sammen med x-verdier. Når de bruker *linspace*-operatoren så vil nemlig de ulike x-verdiene bli lagret som en liste som igjen er en form for datastruktur. I linje 4 snakker E2 om «løkken» som kan knyttes til programmeringsobjektet *løkker*.

5.0 Diskusjon

I dette kapittelet vil jeg diskutere mine funn opp mot relevant teori. Problemstillingen min for denne studien er: *Programmering som en matematisk representasjon*. For å belyse denne problemstillingen har jeg tatt for meg følgende forskningsspørsmål:

1. *Hvordan kan programkode betraktes som en matematisk representasjon?*
2. *Hvordan kan programmering støtte opp under elevers matematiske verbale representasjon?*

I kapittelet vil jeg diskutere de funnene som ble presentert i kapittel 4. For hvert funn vil jeg drøfte mulige årsaker og implikasjoner, og trekke inn relevant teori. Jeg vil også inkludere egne tanker og refleksjoner i diskusjonen. Funnene vil bli diskutert i samme rekkefølge som de ble presentert i kapittel 4. Jeg vil også inkludere et delkapittel hvor jeg drøfter funnene opp mot problemstillingen. Som avslutning på diskusjonskapittelet vil jeg i det siste delkapittelet reflektere kort over studiens styrker og svakheter.

5.1 Programkode kan behandles som noe manipulativt

I delkapittel 4.1 fant jeg at programkode kan behandles som noe manipulativt. Den manipulative representasjonen inngår som én av de fem representasjonsformene vi vanligvis omtaler i matematikken. Som nevnt i teorikapittelet er begrepet «manipulativt» det begrepet som brukes i Lesh et al. (1987, s. 2) sin representasjonsmodell. Det finnes også andre begreper som tilsvarer den manipulative representasjonsformen, som for eksempel NCTM (2014) sitt begrep «physical» som kan oversettes til «fysisk», og Udir (u.å.a) sitt begrep «konkreter». Jeg har valgt å bruke begrepet «manipulasjoner», som skal vise til noe som er manipulerbart. «Manipulasjoner» må ikke nødvendigvis være tilknyttet fysiske objekter.

En mulig forklaring til dette funnet er at manipulative objekter ikke nødvendigvis må være fysiske. Sarama og Clements (2009) snakker om at datamaskiner gir oss tilgang til virtuelle manipulasjoner. Her kan det trekkes linjer til funnet jeg har gjort, hvor vi muligens kan anse programkode som en form for noe som er virtuelt manipulerbart. Funnet mitt viser at programkode som en virtuell manipulativ deler noen likhetstrekk med fysiske konkreter. Et vanlig eksempel på en fysisk manipulativ representasjon er brikker som kan brukes for å representere heltall. På engelsk omtales disse gjerne som «algebra tiles» (NCTM, 2014, s. 78). Disse brikkene kan elevene fysisk ta på og flytte på, for å eksempelvis representere addisjon eller subtraksjon. Den fysiske operasjonen av å ta på og flytte på brikker, kan gjenspeiles i

programkode hvor elevene kan kopiere og lime inn linjer med programkode. Denne prosessen ble også uttrykt verbalt i utdraget jeg presenterte til funnet:

E1: *Nei hvis du går litt ned, også* ***kopier den der****, nei* ***hele den****, også med h'en, også* ***kopier den****, også* ***lim den inn under****. Fordi at der har vi jo for over grafen, også må vi ha en for under grafen, også må vi plusse de to sammen og dele de på to.*

I dette utsagnet har jeg markert noen deler av teksten, som jeg mener kan sammenlignes med det å gjøre fysiske flyttinger av brikker. Utsagnet eksemplifiserer hvordan fysiske manipulasjoner av brikker har blitt erstattet av å kopiere og lime inn virtuelle linjer med programkode, som kan brukes for å lage et program som tilnærmer arealet under en graf ved bruk av trapesmetoden.

Dette funnet viser at tekstbasert programkode kan brukes som en form for virtuell manipulasjon for å representere et matematisk objekt. Programkoden kan manipuleres ved at biter med kode kan kopiers og limes inn ulike steder. Som nevnt i teoridelen blir koblingen mellom programmering og virtuelle manipulasjoner undersøkt hos Goldenberg et al. (2021). Hvor sammenlignbare deres observasjoner er opp mot mitt funn, er usikkert. For det første er studien til Goldenberg et al. (2021, s. 63) fortsatt pågående, noe som gjør at vi ikke kan si noe endelig om observasjonene de har gjort. For det andre så undersøker studien barneskoleelevers arbeid med det blokkbaserte programmeringsspråket «Snap!», mens jeg undersøker videregående elevers arbeid med det tekstbaserte programmeringsspråket Python. Samtidig har programmering mange fundamentale likheter, uavhengig av hvilket språk som brukes. Et eksempel på dette er hvordan et program gir en umiddelbar tilbakemelding til den som skriver kode. For eksempel hvis programkoden kopieres inn i et nytt programmeringsvindu, må det tas hensyn til at variabler fra det «forrige» programmet må defineres på nytt i det nye programmet. Hvis dette ikke blir gjort vil elevene fort få en feilmelding av programmet fordi det ikke har nødvendig informasjon til å kjøre programmet. Goldenberg et al. (2021, s. 62) forteller at denne umiddelbare tilbakemeldingen som programmering tilbyr, gjør det til en mer effektiv manipulativ, kontra fysiske manipulative konkreter som *ikke* gir tilbakemeldinger. Funnet mitt knyttet til manipulativ programkode viser også muligens at forskningen til Goldenberg et al. (2021) kan utvides til videregående skoler som bruker et tekstbasert programmeringsspråk.

I arbeid med den manipulative representasjonen er det viktig at vi er bevisst over hva som er det matematiske objektet som representeres. Funnet jeg presenterer gir en indikasjon på at programkoden representerer det matematiske objektet *rektangler*. Rektanglene som blir

presentert i programkoden kan deretter adderes og divideres for å danne trapeser. Samtidig beregner dette programmet en sum som brukes til å si noe om arealet under en graf. Det er vanskelig å si akkurat hva som blir representert for eleven. Sarama og Clements (2009, s. 146) sier at «good manipulatives are those that aid students in building, strengthening, and connecting various representations of mathematical ideas». Hvorvidt programkode er en *god* form for manipulativ representasjon, kan derimot diskuteres. Basert på det ene funnet jeg har gjort tilknyttet dette kan det ikke trekkes noen konklusjoner. Det viser derimot til et interessant aspekt av hvordan tekstbasert programkode kan oppfattes som en manipulativ representasjon.

5.2 Forståelse av symboler i Python

I delkapittel 4.2 fant jeg at elever ikke nødvendigvis forstår hvordan enkelte symboler ble brukt i programmeringsspråket Python. Her viste jeg til et utdrag hvor elever stilte spørsmål knyttet til symbolene «=» og «+=», og det ga meg inntrykket av at elevene kanskje ikke forsto helt hvordan disse tegnene ble brukt i programkoden. Utsagnet fra den ene eleven var «*Det er liksom det som er problemet med fra matte som jeg skriver på papiret mitt*», hvor jeg tolker dette som at eleven uttrykker at det å gjøre matematikk med programmering ikke er som å gjøre det «på papir».

For å drøfte en mulig årsak til dette funnet vil jeg trekke frem artikkelen til Bråting og Kilhamn (2021) hvor de undersøker sammenhenger mellom algebra og programmering. En av situasjonene de ser på i sin artikkel er bruken av «$a+= 1$» eller «$a = a + 1$» i programmeringsspråket JavaScript. Dette er notasjoner som også brukes i programmeringsspråket Python. Et av funnene deres er disse symbolene «+=» og «=» har en annen betydning i programmering enn i det det har i vanlig algebra. De konkluderer med følgende: «Therefore, conversions between natural language and symbolic registers will be different if the conversion is to JavaScript or to algebraic notation. In addition, a conversion between JavaScript and algebra entails yet another a non-congruent transformation» (Bråting & Kilhamn, 2021, s. 180). I dette sitatet har de brukt begrepet «conversions» fra Duval (2006). Dette begrepet er også noe jeg har inkludert i teorikapittelet, hvor jeg har gitt det den norske oversettelsen «overgang». Altså sier Bråting og Kilhamn (2021) at *overgangen* fra JavaScript til algebra ikke er en kongruent transformasjon. Dette betyr at symbolene ikke er direkte overførbare mellom algebra og JavaScript, eller i dette tilfellet til Python.

Konklusjonen til Bråting og Kilhamn (2021) kan også knyttes opp mot mitt funn. Utsagnet fra den ene eleven var: «*Det er liksom det som er problemet med fra matte som jeg skriver på*

papiret mitt», hvor eleven uttrykker at det å gjøre matematikk med programmering ikke er som å gjøre det «på papir». Her mener jeg at «på papir» kan tolkes som «med algebraiske notasjoner». Programmeringsspråk bruker mange av de samme symbolene som vi er vant til fra når vi jobber med algebraiske notasjoner. Men siden overgangene ikke er kongruente kan det medføre til at elever ikke nødvendigvis forstår symbolbruk i Python.

En annen årsak til dette funnet er nok også at programmeringsnivået i klassen var spredt. Det kan dermed være at de elevene som stilte spørsmål til symbolbruk i Python, kanskje ikke hadde så mye erfaring med programmering. Da blir det fort vanskelig å forstå hvorfor kjente symboler fra algebra brukes på en annerledes måte i Python. I utdraget ser vi at eleven forklarer at «$A+= firkant$» er det samme som å skrive «$A = A + firkant$». I vanlig matematikk ville en slik type notasjon vært en matematisk umulighet, og det ville ikke vært hensiktsmessig å uttrykke det med algebraiske notasjoner. Det å starte med å skrive «$areal = 0$» er matematisk korrekt, men i arbeid med numerisk integrasjon ville en ikke startet med å skrive slik. Disse uttrykkene er vanlig å bruke i programmering, og lite programmeringserfaring kan derfor resultere i uvisshet til bruk av disse symbolene.

Dette funnet bidrar til å si noe om programmering og programkode som en egen form for matematisk representasjon. Duval (2006, s. 110) sier at et tegn kun får mening når det er innenfor sitt eget semiotiske system. Med andre ord må vi se på programmering og algebra som to forskjellige semiotiske systemer, hvor de samme symbolene får en ulik mening. Dette lufter spørsmålet om vi kan se på programmering som et eget register. Til funnet kan det bli gjort observasjoner av at det kan gjøres *behandlinger* av tegn i programmering. I utdraget forteller eleven at «$A+= firkant$» er det samme som å skrive «$A = A + firkant$». Her gjør eleven en behandling av semiotiske representasjoner. Behandlinger var den ene formen for transformasjon som blir beskrevet hos Duval (2006). Samtidig sier Duval (2006, s. 111) at «Not all semiotic systems are registers, only the ones that permit a transformation of representation». Fra funnet jeg har gjort kan det altså dannes et argument som tilsier at programkode kan anses som et eget register, i og med at det tillater transformasjonen *behandling*.

En annen implikasjon av funnet er at det kan være nyttig å praktisere hvordan man verbaliserer symboler i programmering. Bråting og Kilhamn (2021, s. 180) sier for eksempel at symbolene «$a = a + 1$» kan uttrykkes verbalt som «legg til 1 til verdien a». Dette vil være forskjellig fra å si «a er lik a pluss 1» som ville vært korrekt fra et algebraisk ståsted. Hana (2013, s. 161-165) forteller om verbalisering av matematiske notasjoner, og sier at et uttrykk kan verbaliseres på

flere måter. Dette er noe som kan være viktig å ha i bakhodet når man arbeider med programmering. Hana (2013, s. 162) foreslår en øvelse for å øve på å verbalisere symbolske uttrykk. Øvelsen går ut på at elevene skal arbeide i par, hvor den ene eleven skal lese opp et symbolsk uttrykk, og den andre eleven skal skrive ned uttrykket slik som det blir sagt. Denne øvelsen kan være et forslag til et didaktisk opplegg i undervisning av programmering for å unngå misoppfatninger mellom symboler i programmering og i algebra. Verbalisering av programmeringssymboler er også noe jeg som lærer kommer til å tenke over hvis jeg for eksempel skal forklare en programkode foran elever.

5.3 Elever skriver programkode for seg selv

I delkapittel 4.3 fant jeg at elevene hadde en tendens til å skrive kode for seg selv. Som nevnt i metodekapittelet ble det før gjennomføringen av datainnsamlingen gitt en regel om at elevgruppene skulle arbeide på én PC per gruppe. Videre kaller jeg denne regelen for «én pc per gruppe». Denne regelen var inspirert av Liljedahl (2021) sin teori om «Building Thinking Classrooms», som går ut på at elevgrupper skal dele én tusj på én tavle. Motivasjonen for å benytte denne regelen var for å øke elevenes verbale aktivitet, samtidig som det ble mer praktisk å få samlet all data fra en gruppe på ett opptak.

Basert på mine observasjoner og inntrykk vil jeg påstå at «én PC per gruppe» hadde en positiv effekt i måten det fikk elevene til å samhandle på. Det ble mye dialog på de ulike gruppene, og opptakene gjorde det enkelt for meg å sette meg inn i elevenes kontekst. Samtidig observerte jeg også at «én PC per gruppe» kunne føre til at det ofte var stille på gruppen mens én elev programmerte. Det som gjorde dette interessant var at det skjedde på flere av gruppene.

En mulig årsak til dette kan være knyttet til spørsmålet om programmering som en indre representasjon. Når man driver med programmering så har man gjerne en idé «i hodet» om hvordan programkoden skal se ut, eller om hvordan programmet skal fungere. Disse ideene kan være vanskelig å uttrykke verbalt for resten av gruppen. Dermed blir det lettere å jobbe «en-til-en» med PC-en, hvor en selv kan fokusere på å få skrevet ned og testet ideene sine. Her vil også jeg trekke inn en personlig kommentar og si at jeg selv har erfart lignende situasjoner. Programmering er en aktivitet som jeg personlig foretrekker å gjøre alene, da jeg ofte har en egen idé om hvordan programmet skal fungere. Et annet aspekt ved denne årsaken kan knyttes opp til det Goldenberg et al. (2021, s. 62) sier om at programmering er en «live notation». Med dette menes at programmering er en interaktiv aktivitet, hvor den som programmerer vil aktivt få tilbakemeldinger fra programmet det kodes i. Det vil si at man kan skrive en programkode,

trykke «kjør»-knappen, og nesten umiddelbart få en tilbakemelding på datamaskinen. Igjen av personlig erfaring er denne «input»-«output» interaksjonen noe som kan kreve en del prøving og feiling, noe som kan være enklere å håndtere når man skriver kode for seg selv. Kan tenkes at vi foretrekker å «snakke» til datamaskinen fremfor å snakke med medstudenter når vi arbeider med programmering?

En annen mulig årsak kan være at elevene ikke var vant til å jobbe med programmeringsoppgaver i grupper. Undervisningsøkten var lagt opp slik at elevene skulle arbeide på én felles PC. Denne måten å arbeide med programmering på var muligens noe elevene ikke hadde mye erfaring med tidligere. De fleste jobber nok vanligvis med programmering på sin egen PC, og har et eget system på hvordan de foretrekker å skrive programkode. Samtidig var det også ulikt programmeringsnivå mellom de ulike informantelevene. Dermed kan det være vanskelig å uttrykke seg verbalt hvis en ikke har god kontroll på hvordan programmering fungerer.

Dette funnet har muligens implikasjoner som sier noe om programmering som en indre representasjon. I teorikapittelet henviste jeg til Goldin (2020) som forteller at representasjoner kan være både interne og eksterne. Av disse to representasjonene er det kun de eksterne representasjonene som kan observeres. Jeg vil i drøfting av dette funnet stille et spørsmål om hvorvidt programmering er en prosess som baserer seg på indre representasjoner. Observasjonene jeg har gjort til dette funnet sier oss kanskje at programmering tilrettelegger for mer bruk av indre representasjoner, noe som medfører at elevene har en tendens til å programmere for seg selv. Som Goldin (2020, s. 567) sier kan indre representasjoner referere til elevers kinestetiske koding av operasjoner og deres heuristiske planer og strategier for problemløsning. Slik som Sevik (2016, s. 9) definerer programmering er både forståelsen av operasjoner og strategier for problemløsning sentrale elementer. Her kan det også trekkes tråder til problemløsning og abstraksjoner knyttet til algoritmisk tenkning slik det blir definert hos Udir (2019b). Indre representasjoner kan også knyttes til hvordan vi visualiserer når vi programmerer. Arcavi (2003) snakker om visualiseringens rolle i matematikken. Han forteller at visualisering blant annet handler om å kunne danne seg kognitive forestillinger av matematiske objekter. Funnet mitt om at elevene til tider er stille og programmerer for seg selv kan ha en tilknytning til algoritmisk tenkning og den kognitive visualiseringen, som igjen sier noe om hvordan programmeringen og programkoden prosesseres mental som en indre representasjon.

På en annen side kan det argumenters for at det blir feil å sette fokus på at programmering må være knyttet til indre representasjoner, uten å sette det opp mot eksterne representasjoner. Indre og eksterne representasjoner kan nemlig sees på som overlappende. Lesh et al. (1987, s. 1) snakker for eksempel om eksterne representasjoner som en legemliggjøring av indre representasjoner, hvor alle eksterne representasjoner må ha utgangspunkt i en indre representasjon. Indre representasjoner er heller ikke noe som kan observeres direkte, noe som gjør det vanskelig å si noe om. Funnet mitt impliserer derimot at det kan virke som om den prosessen som inngår i å skrive en programkode tilrettelegger for bruk av indre representasjoner.

Til slutt vil jeg koble dette funnet mitt opp mot observasjoner som blir gjort i artikkelen til Goldenberg et al. (2021, s. 50). En av observasjonene som blir presentert i artikkelen er at programmering kan bidra til å bygge bedre indre representasjoner, som vil gjøre matematiske objekter mer tilgjengelige. Denne observasjonen blir bare formidlet som en formodning i artikkelen, da de ikke har noe underbyggende bevis. Jeg synes det er interessant at jeg har gjort et lignende funn i min studie. Dette indikerer muligens at det er noe mer som ligger bak at elever har en tendens til å skrive kode for seg selv i gruppearbeid med matematiske programmeringsoppgaver.

5.4 Programmeringsobjekter og programmeringsbegreper kan mediere elevenes verbale representasjoner av matematiske objekter

I delkapittel 4.4 fant jeg at programmeringsobjekter og programmeringsbegreper kan mediere elevenes verbale representasjoner av matematiske objekter. Dette handler om at programmeringsobjekter og programmeringsbegreper har en innvirkning på elevenes dialog omkring de matematiske objektene.

En mulig årsak til dette funnet er hvordan programmeringsobjekter og programmeringsbegreper er vanskelig å unngå i arbeid med programmering. Programmeringsobjektene presenterte jeg i teorikapittelet, og består av *rekkefølger*, *løkker*, *betingelser*, *operatorer* og *datastrukturer*. Av disse var det spesielt programmeringsobjektene *rekkefølger*, *løkker* og *datastrukturer* som fikk ekstra mye oppmerksomhet i mye av elevdiskusjonene. Hvor ulike variabler og operatorer skal plasseres i programmet og hvordan de skal lagre eller oppdatere variabler virker som å være noe elevene tenker over når de arbeider med programmering. Gazoni (2018) forteller at en forskjell mellom programmeringsspråk og naturlig språk er knyttet til grad av vaghet. Vårt naturlig språk er vagt, hvor ord og uttrykk kan

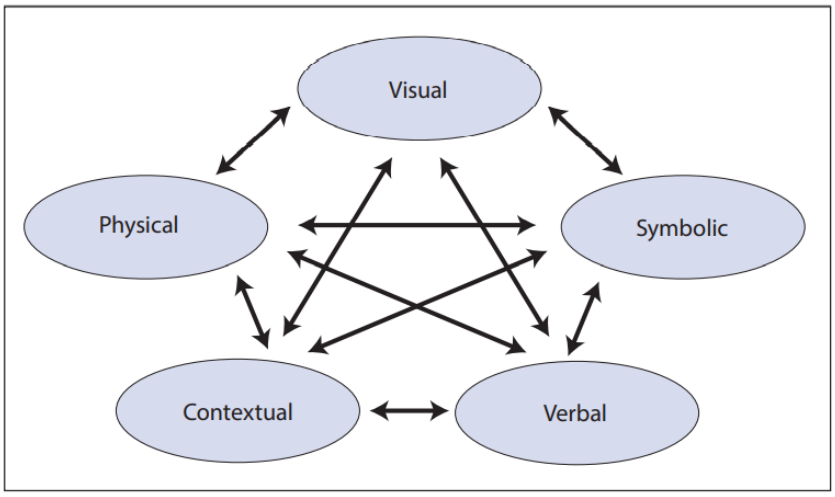
tolkes på mange forskjellige måter. Programmeringsprogram krever derimot presis språkbruk slik at det kan lese av datamaskinens prosessorer. Det kan derfor være krevende for elevene å uttrykke seg verbalt i arbeid med programmering. De ulike programmeringsobjektene må forstås og tas hensyn til når en diskuterer en programmeringsoppgave. Wallace (2004) bruker begrepet «å verbalisere» for å beskrive tankeprosessen når en skal omgjøre mellom ulike språk. I dette funnet kan denne omgjøringen innebære å gå fra programmeringsspråk til naturlig språk. I noen tilfeller har programmeringsspråket mange likheter med algebraiske notasjoner, noe som elevene er vant til å verbalisere. I andre tilfeller er det få likheter med algebraiske notasjoner, og da kan programmeringsobjekter og programmeringsbegreper mediere elevenes verbale representasjoner, slik at det lettere kan kommuniseres til resten av elevgruppen.

En av implikasjonene til dette funnet er at det bidrar til å si noe om programmering som et eget register innenfor semiotiske representasjoner. I delkapittel 5.2 diskuterte jeg funnet om symboler i programmering, og viste til en elev som gjorde en transformasjon i form av behandling. Til dette funnet gjør elevene også *overganger* mellom semiotiske representasjoner. Overganger er ifølge Duval (2006) en transformasjon som innebærer en endring av register. Verbalt språk er et av de eksemplene Duval (2006) trekker inn som et eget register. Med andre ord, når elevene bruker programmeringsobjekter og programmeringsbegreper i sitt verbale register, så må de ha gjort en overgang fra et sted. Her kan det argumenteres for at denne overgangen blir gjort fra programmering til verbalt språk, noe som tilsier at programmering er et eget register. Funnet knyttet til elevers verbale diskusjoner hjelper oss derfor til å belyse et nytt aspekt av hvordan programmering og programkode kan betraktes som en matematisk representasjon.

5.5 Programmering som en matematisk representasjon

Så langt i diskusjonskapittelet har jeg diskutert funnene som ble presentert i kapittel 4. Diskusjonen har vært strukturert etter å drøfte mulige årsaker og implikasjoner til funnet. Nå ønsker jeg å drøfte funnene opp mot problemstillingen: *Programmering som en matematisk representasjon.*

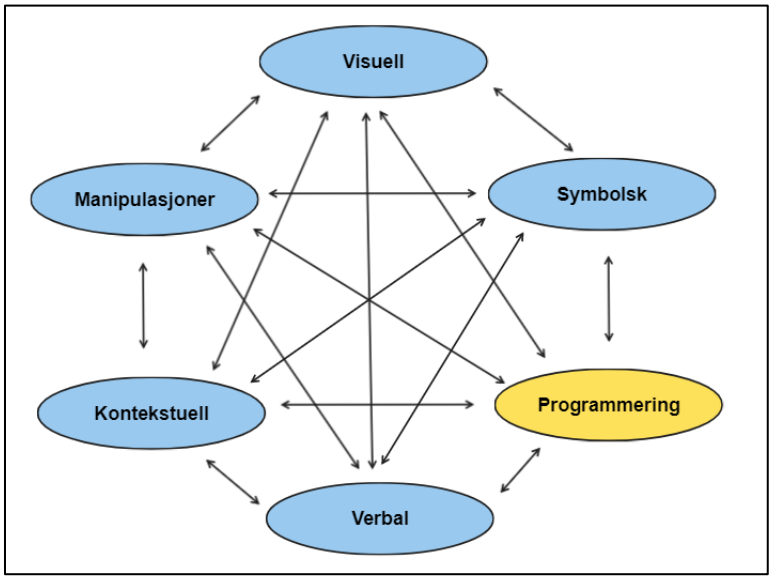
I teoridelen la jeg frem figur 1, som viser til følgende modell fra NCTM (2014):



Figur 1. Ulike typer representasjoner. Hentet fra NCTM (2014, s. 25).

Modellen viser til ulike representasjonsformer vi benytter oss av i arbeid med matematikk. Disse ulike formene oversetter jeg til *visuell, symbolsk, verbal, kontekstuell* og *fysiske.* I drøftingen som følger ønsker jeg å ta utgangspunkt i denne modellen. Hensikten vil være å belyse temaet programmering som en matematisk representasjon, ved å knytte det opp mot allerede etablerte representasjonsformer.

På bakgrunn av mine funn og min diskusjon, vil jeg drøfte en tanke om å utvide modellen til NCTM (2014). I figur 11 nedenfor presenterer jeg et forslag til hvordan denne modellen kanskje kan utvides.



Figur 11. Min modifisering av modellen til NCTM (2014).

I min utvidete versjon av modellen har jeg gjort noen endringer fra den originale modellen, som må kommenteres. For det første har jeg oversatt modellen til norsk, for at den skal stå mer i stil

til denne oppgaven, som også er skrevet på norsk. For det andre så har jeg endret navnet på boblen «physical» til norske «manipulasjoner». Dette har jeg gjort på bakgrunn av diskusjonen i delkapittel 5.1, hvor jeg diskuterer funnet knyttet til programkode som noe manipulativt. I diskusjonen trakk jeg inn teori som foreslo at manipulasjoner både kunne bli gjort fysisk eller virtuelt. Boblen «manipulasjoner» er derfor ment til å dekke både manipulering av fysiske konkreter, og av noe virtuelt manipulerbart. Som nevnt tidligere bruker Lesh et al. (1987) begrepet «manipulative models» i deres representasjonsmodell, men i beskrivelsen nevner de ikke noe om hvorvidt virtuelle manipulasjoner inngår i denne representasjonsformen.

Den viktigste endringen i min utvidete versjon, er tilsettingen av den nye boblen «programmering». Programmerings-boblen har i min modell fått en egen farge for å illustrere at den er et nytt tilskudd. Jeg påstår ikke at denne tilsettingen av en ny representasjonsform kan anses som korrekt, og at «slik er det». Modellen er laget for å lufte spørsmålet om hvorvidt programmering kan anses som en egen representasjonsform sammen med de allerede etablerte representasjonsformene. Pilene mellom boblene i modellen er ment for å illustrere at man skal kunne gjøre translasjoner mellom de ulike representasjonsformene. En av de tingene som derfor må undersøkes er hvorvidt programmerings-representasjonen kan oversettes til de andre representasjonsformene. Funnene jeg har gjort, og diskusjonen rundt funnene, bidrar til å si noe om programmering i henhold til de andre representasjonsformene. Dette er noe jeg ønsker å drøfte i avsnittene som følger. Jeg kommer til å drøfte programmering knyttet til manipulasjoner, verbal og symbolsk representasjonsform. Jeg kommer ikke til å drøfte programmering knyttet til den visuelle og kontekstuelle representasjonsformen, da jeg ikke har gjort funn eller diskusjoner tilknyttet disse.

I delkapittel 5.1 diskuterte jeg funnet om at programkode kan brukes til å utføre en form for virtuelle manipulasjoner. Programkoden står her sentralt da det var den som ble betraktet som noe manipulativt. Tidligere har jeg beskrevet programkode som en innebygd del av programmering. Derfor kan det også sies at programkode vil være noe som inngår i boblen programmering. I diskusjonen rundt dette funnet trekker jeg inn artikkelen til Goldenberg et al. (2021) som også undersøker programmering som en virtuell manipulasjon. Hovedforskjellene mellom denne artikkelen og min studie er som nevnt at de ser på barneskoleelever med et blokkbasert programmeringsspråk, og jeg ser på videregående elever med et tekstbasert programmeringsspråk. Derfor kan det stilles spørsmål til hvor sammenlignbart mitt funn er opp mot denne forskningen. Samtidig er dette forskning som fremdeles er pågående, og det kan derfor ikke trekkes en endelig vurdering av hvor valid translasjonen mellom programmerings-

representasjonen og den manipulative representasjonen er. Jeg har i mitt funn observert at eleven behandler programkoden som noe manipulativt i måten hen operer på i den observerte situasjonen. Fra et representasjonsperspektiv gjenstår det blant annet et ubesvart spørsmål knyttet til hva som skjer med det matematiske objektet i denne situasjonen. *Hva* ser eleven i denne manipulasjonsprosessen? Dette er spørsmål jeg ikke har empiri til å svare på. Det kan altså virke som om programkoden som en del av programmerings-representasjonen, kan oversettes til manipulasjoner, men hvor sterk denne tilknytningen er blir det vanskelig å si noe om.

Translasjonen mellom programmering og den verbale representasjonsformen er noe jeg har diskutert i delkapittel 5.4, hvor jeg ser på funnet av at programmeringsobjekter og programmeringsbegreper kan mediere elevenes verbale representasjoner av matematiske objekter. Denne overgangen fra programmering til verbale uttrykk kommer også til uttrykk i utdragene som blir presentert i funn om manipulativ programkode, og om symboler i Python. Jeg observerte og under datainnsamlingen at det var mye verbal aktivitet i gruppene, dette gjenspeiler seg også i transkripsjonene. Men mye verbal aktivitet betyr ikke nødvendigvis at det er mange matematiske verbale representasjoner. Det er tross alt det matematiske objektet som må kunne gjenkjennes i de ulike representasjonene.

Det jeg anser som det mest uavklarte forslaget i min modifiserte representasjonsmodell er at programmeringsboblen står adskilt fra den symbolske boblen. Programkode som produseres i programmeringsspråket Python består av ulike symboler fra ASCII-tegnsettet. Dette medfører at programkode blir representert på en symbolsk form, på samme måte som man i algebra bruker algebraiske symboler for å representere matematiske objekter. Herfra kan vi argumentere for at programmering er symbolspråk, noe som muligens tilsier at programmerings-representasjonen bør være innebygd i den symbolske representasjonen. Men aspekter av diskusjonen i delkapittel 5.2, om at elever nødvendigvis ikke forstår symboler i Python, kan kanskje vise til at symbolene i seg selv ikke er nok til å si at det er en symbolsk representasjonsform.

I delkapittel 5.2 diskuterte jeg funnet av at elever nødvendigvis ikke forstår symboler i Python. Det er mulig at den diskusjonen bidrar til å sette programmering i sin egen boble. I den diskusjonen trakk jeg inn artikkelen til Bråting og Kilhamn (2021) som undersøker symboler i programmering opp mot algebraiske notasjoner. I deres artikkel konkluderer de med at symboler som er like i algebra og i programmeringsspråk, ikke er direkte overførbare. Her snakkes det om symbolene «=» og «+=», som også er de symbolene jeg gjorde meg

oppmerksom på i mitt funn. I denne diskusjonen viste jeg også til at en elev gjorde behandlinger av symbolet «+=», hvor «$A+= firkant$» ble sagt å være det samme som å skrive «$A = A + firkant$». Jeg la frem et argument om at programkode kunne bli ansett som et eget register. Programmering som eget register ble også diskutert i delkapittel 5.4 om funnet med verbale representasjoner, hvor jeg argumenterte for at elever gjorde overganger fra programmering til verbal representasjon.

Symbolene som blir brukt i programmering må forstås på en annen måte enn hvordan de gjør i for eksempel et algebraisk register. Samtidig må disse symbolene gjerne underbygges i form av programmeringsobjekter og programmeringsbegreper. Funnet som blir diskutert i delkapittel 5.3 om elever som skriver programkode for seg selv gir også en mulig indikasjon at elevers mentale prosessering er annerledes i arbeid med programmering. Observasjoner jeg har gjort her kan også sammenlignes med noen av de observasjonene som har blitt gjort hos Goldenberg et al. (2021) sin pågående forskning. Det kan derfor tenkes at hvordan vi oppfatter disse symbolene i programmering tilrettelegger for en annen type mental prosessering, enn med andre symboler. Her vil jeg trekke inn Sfard (1991) sine to typer av forståelse: strukturell og operasjonell forståelse. Sfard (1991) forteller den strukturelle og operasjonelle forståelsen ofte viser seg i de ulike representasjonene vi bruker når vi prosesserer kunnskap mentalt. I delkapittel 2.3.3, om strukturell og operasjonell forståelse, viste jeg til en figur fra Sfard (1991, s. 6) som eksemplifiserer at programkode må forstås på en operasjonell måte. Den operasjonelle forståelsen kan ses på som dynamisk, sekvensiell og detaljert, og appellerer derfor til programmeringsrepresentasjonen. Programmering har også den unike egenskapen at vi alltid må ta hensyn til at det først og fremst er PC-en som skal «forstå» programkoden. Dette er noe vi må ta i betraktning når vi skriver symboler, og tolker symboler innenfor programmering. Selv om programkode består av symboler, blir disse symbolene representert i en unik programmerings-kontekst, som bidrar til å endre hvordan vi forstår og internaliserer disse symbolene.

Jeg vil igjen si at den modifiserte modellen jeg har presentert har som hensikt å illustrere et forslag til en utvidelse av de etablerte representasjonsformene vi vanligvis omtaler i matematikken. Modellen kan også bidra til å danne et grunnlag for videre drøfting av programmering som en matematisk representasjon. Jeg har diskutert ulike aspekter av hvordan programkode kan anses som en matematisk representasjon, og jeg har sett på hvordan programmering støtter opp under elevens matematiske verbale representasjoner. Å undersøke

dette har bidratt til å belyse ulike aspekter av programmering som en matematisk representasjons som jeg har drøftet i dette delkapittelet.

5.6 Studiens styrker og svakheter

I dette delkapittelet vil jeg gjøre en kort refleksjon over studiens styrker og svakheter. I denne studien har jeg benyttet meg av et kvalitativt forskningsdesign, i form av en hermeneutisk-fenomenologisk casestudie, for å undersøke programmering som en matematisk representasjon. Studien har blitt gjennomført i R2-klasse, med et utvalg på 6 elevgrupper med 3-4 elever per gruppe. Datainnsamlingen ble gjort som en del av en undervisningsøkt, hvor elevene arbeidet med oppgaver tilknyttet numerisk integrasjon.

Jeg vil starte med å reflektere over de metodiske valgene for studien. Jeg vil starte med å si at jeg er fornøyd med hvordan undervisningsøkten og datainnsamlingen gikk for seg. Selv om jeg hadde rolle som forsker i undervisningsøkten, følte jeg meg ikke som fremmed for elevene. Informantelevene arbeidet bra, og med lyd-, skjerm- og videoopptak som metode fikk jeg samlet inn mye data. Opptakene ga meg god innsikt i elevenes kontekst når jeg observerte og analyserte opptakene. Fra et hermeneutisk-fenomenologisk perspektiv har dette derfor styrket muligheten til å undersøke elevenes forståelse av et fenomen. Kvaliteten og omfanget av datamaterialet anser jeg som en av studiens styrker.

Størrelsen på utvalget kan derimot sees på som en svakhet ved studien. Jeg har kun fått samlet inn data fra én klasse i én undervisningsøkt. Dersom jeg hadde fått opptak fra flere klasser, eller fra flere undervisningsøkter, kunne jeg fått et datamateriale basert på flere ulike elevperspektiver. Som igjen kunne gitt meg et bredere grunnlag for å undersøke problemstillingen. Størrelsen på utvalget begrenser også i hvilken grad resultatene er generaliserbare, men dette har heller ikke vært min intensjon siden jeg har valgt å bruke et kvalitativt forskningsdesign.

Jeg vil og rette en kritisk refleksjon til oppgaven som ble gitt til elevene. Så å si alle oppgavene gitt ut på at elevene skulle skrive en programkode. Når jeg ser tilbake på det, kunne det vært interessant å gi elevene en oppgave som går ut på å tolke en programkode. Her kunne jeg på forhånd ha utformet en eller flere programkoder som løste ulike matematiske oppgaver. Elevenes oppgave ville da vært å finne ut hva programmet gjør. Dette kunne muligens gitt interessante data på hvordan matematiske objekter blir representert i en programkode. Hadde datainnsamlingen blitt gjort i flere undervisningsøkter kunne jeg også ha gitt oppgaver knyttet til andre matematiske tema.

6.0 Avslutning

I denne masteroppgaven har jeg undersøkt problemstillingen *programmering som en matematisk representasjon*, med forskningsspørsmålene:

Forskningsspørsmål 1: *Hvordan kan programkode betraktes som en matematisk representasjon?*

Forskningsspørsmål 2: *Hvordan kan programmering støtte opp under elevers matematiske verbale representasjon?*

Problemstillingen og forskningsspørsmålene har blitt undersøkt ved å analysere data i form av lyd-, skjerm- og videoopptak fra R2 elever som arbeider med matematiske programmeringsoppgaver om numerisk integrasjon.

I dette avslutningskapittelet vil jeg starte med å fremlegge en konklusjon til forskningsspørsmålene og problemstillingen. Deretter vil jeg se på teoretiske og praktiske implikasjoner av studien jeg har gjennomført. Til slutt vil jeg fortelle kort om veien videre.

6.1 Konklusjon

Analysen som ble gjort av datamaterialet mitt resulterte i de funnene som jeg har presentert i kapittel 4. Disse funnene ble videre diskutert i kapittel 5. På bakgrunn av disse funnene, og diskusjonen jeg har foretatt meg, vil jeg først trekke konklusjoner til de to forskningsspørsmålene jeg har tatt for meg i oppgaven. Til slutt vil jeg forklare hvordan disse konklusjonene belyser problemstillingen *programmering som en matematisk representasjon.*

Forskningsspørsmål 1: Hvordan kan programkode betraktes som en matematisk representasjon?

Programkode kan betraktes som en matematisk representasjon da det kan tilrettelegge for at elever utfører virtuelle manipulasjoner av matematiske objekter. Programkode kan inneholde matematiske symboler som elever må forstå på en annerledes måte enn i vanlig algebraisk sammenheng. Disse symbolene må tolkes innenfor et eget semiotisk system knyttet til programmering. Av denne grunn kan programkode anses som et eget register av semiotiske representasjoner. Elever har en tendens til å skrive programkode for seg selv, noe som kan indikere at prosessen som inngår i å skrive en programkode tilrettelegger for mer bruk av indre representasjoner av matematiske objekter.

Forskningsspørsmål 2: Hvordan kan programmering støtte opp under elevers matematiske verbale representasjon?

Programmering støtter opp under elevers matematiske verbale representasjon ved at programmeringsobjekter og programmeringsbegreper kan mediere elevenes dialog. Det «å mediere» betyr i denne sammenhengen at bruk av programmeringsobjekter og programmeringsbegreper har en innvirkning på hvordan elevene snakker, beskriver eller uttrykker seg om de matematiske objektene. Programmering kan på denne måten bidra med å skape en bro mellom den verbale representasjonen og det matematiske objektet.

Når elever snakker om matematiske objekter i arbeid med programmering tar de ofte hensyn til rekkefølger, datastrukturer eller andre programmeringsobjekter i dialogen.

Programmering som en matematisk representasjon

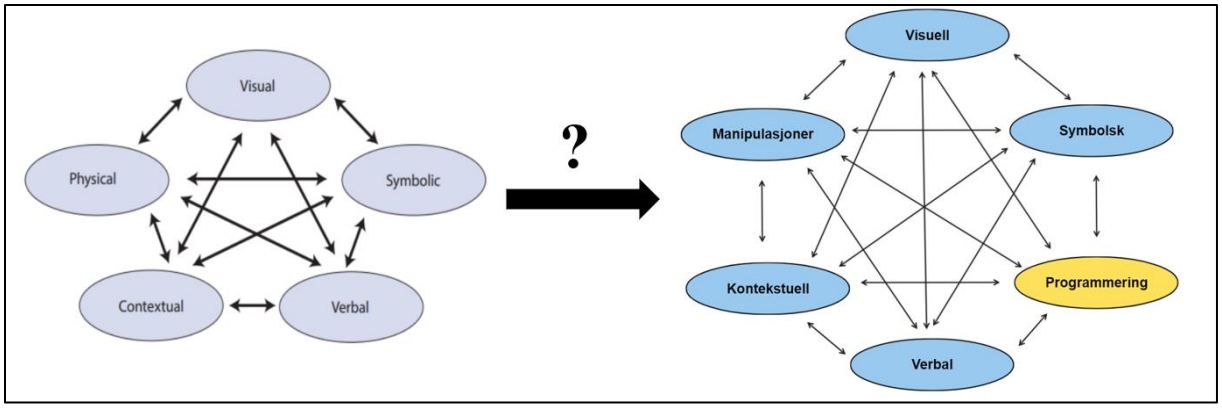
Programmering kan anses som en egen representasjonsform som tillater translasjoner til allerede etablerte representasjonsformer. Arbeid med programmering kan medføre at elever produserer ekstern programkode og verbal dialog som representerer matematiske objekter. Disse eksterne produksjonene må forstås innenfor et eget semiotisk system av representasjoner. Samtidig kan programmering ha en innvirkning på våre indre representasjoner av hvordan vi oppfatter et matematisk objekt.

Alle de tre konklusjonene jeg har gitt ovenfor baserer seg på empiri fra én R2 klasse. Noen av funnene har også blitt basert på enkeltobservasjoner i analysen. Jeg vil derfor ikke påstå at konklusjonene jeg gir er endelige. Uavhengig av dette vil jeg påstå at studien min bidrar til å gi et bredere bilde av programmering som en matematisk representasjon, noe som også kan bidra til å fylle gapet i den eksisterende forskningen.

6.2 Teoretiske og praktiske implikasjoner

Som nevnt innledningsvis er programmering som en matematisk representasjon et felt hvor jeg finner lite tidligere forskning. Derfor mener jeg at denne masteroppgaven kan anses som et bidrag til forskningen på dette området. Matematiske representasjoner og programmering, som temaer i seg selv, er derimot mye utforsket. Jeg har i denne studien ønsket å undersøke disse to sammen. Arbeidet med denne studien har fått meg til å fundere på spørsmålet om hvorvidt programmering kan anses som en egen representasjonsform, på lik linje med de fem representasjonsformene vi vanligvis snakker om i matematikk. Dette var noe jeg diskuterte i delkapittel 5.5, og jeg presenterer det her som en teoretisk implikasjon av studien jeg har

gjennomført. I figur 12 nedenfor viser jeg til både NCTM (2014) sin modell, som jeg har tatt utgangspunkt i, og mitt forslag til en utvidet modell som inneholder representasjonsformen «programmering».



Figur 12. Til venstre; modell hentet fra NCTM (2014, s. 25), til høyre; min utvidelse.

Som sagt er modellen min ment som et forslag, og derav spørsmålstegnet over pilen mellom de to modellene. Før det kan fastslås at programmering kan anses som en egen representasjonsform, må det bli gjort mer forskning på hvordan programmering kan kobles til de andre representasjonsformene. Spesielt viktig mener jeg det vil være å undersøke hvordan programmering kan distanseres fra den symbolske representasjonsformen. Mine funn i denne masteroppgaven kan ikke si noe endelig om programmering som en egen representasjonsform. Samtidig bidrar funnen til å si noe om ulike aspekter ved programmering tilknyttet manipulasjoner, symboler og verbale representasjoner.

Denne studien har også praktiske implikasjoner, sett fra et matematikkdidaktisk perspektiv. Funnene mine viser først og fremst at matematiske objekter er til stede i elevers arbeid med matematiske programmeringsoppgaver. Samtidig viser for eksempel funnet knyttet til elevers verbale representasjoner at programmeringsobjekter og programmeringsbegreper kan mediere elevdialogen. Dette er informasjon som kan overføres til hvordan man som lærer underviser i matematikk ved bruk av programmering. For eksempel bør en lærer være bevisst på hvordan en verbaliserer seg i arbeid med programmering. Det er også viktig å være bevisst på at symboler i programmering kan ha en annerledes betydning en symboler vi arbeider med ellers i matematikk. Metodene jeg har gjennomført i studien kan også bidra til å inspirere med hvordan man kan sette opp en undervisningsøkt hvor elevene arbeider med programmeringsoppgaver i grupper.

6.3 Veien videre

Som sagt har jeg ikke funnet mye tidligere forskning på programmering som en matematisk representasjon. Jeg vil derfor påstå at det finnes mange muligheter for videre forskning på akkurat dette feltet. Samtidig vil jeg også påstå at det *bør* bli gjort mer forskning på dette feltet. Teknologi tar stadig større plass i hverdagslivene våre, og jeg mener det er utrolig viktig at skolen bidrar til at elever tilegner seg teknologiske kunnskaper. Jeg er også positiv til programmering som en del av matematikkfaget, og jeg har troen på at programmering er en god bidragsyter for å styrke elevers matematiske kunnskaper. Igjen bidrar dette til å underbygge behovet for mer undersøkelser knyttet til programmering som en matematisk representasjon.

I min studie ga jeg informantelevene mine en oppgave hvor de skulle programmere og produsere en programkode. Dette valget var påvirket av at gjennomføringen skulle bli gjort som en del av et undervisningsopplegg. Som et forslag til videre forskning hadde det vært interessant å gi en omvendt type oppgave. Altså gi elevene ulike programkoder, og be dem diskutere hva de ser, hva programkoden gjør eller liknende. Jeg tror også det kunne vært interessant å undersøke programmering knyttet til andre matematiske områder enn der hvor det blir nevnt eksplisitt i kompetansemålene. Dette ble også drøftet i delkapittelet om studiens styrker og svakheter. Et annet forsalg kan være å gjøre undersøkelser på et større utvalg elever. Hvor man kan undersøke elever på ulike trinn, elever med ulik programmeringserfaring, elever som har programmering som et eget valgfag, og så videre. Her finnes det mange muligheter.

Jeg kan ikke si noe endelig med de funnene jeg har gjort i denne studien. Jeg kan derimot si med sikkerhet at hele arbeidsprosessen med masteroppgaven har gjort meg, som fremtidig mattelærer, bedre rustet til å bruke programmering og matematiske representasjoner i et matematikklasserom.

Referanseliste

Arcavi, A. (2003). The Role of Visual Representations in the Learning of Mathematics. *Educational Studies in Mathematics*, *52*(3), 215-241. https://doi.org/10.1023/a:1024312321077

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

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

Bryman, A. (2012). *Social Research Methods* (4. utg.). Oxford University Press.

Bråting, K. & Kilhamn, C. (2021). Exploring the intersection of algebraic and computational thinking. *Mathematical Thinking and Learning*, *23*(2), 170-185. https://doi.org/10.1080/10986065.2020.1779012

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

Cobley, P. (2001). *The Routledge Companion to Semiotics and Linguistics*. Psychology Press.

Curtin, B. (2009). *Semiotics and Visual Representation* [Chulalongkorn University]. Bangkok, Thailand.

Duval, R. (2006). A Cognitive Analysis of Problems of Comprehension in a Learning of Mathematics. *Educational Studies in Mathematics*, *61*(1-2), 103-131. https://doi.org/10.1007/s10649-006-0400-z

Enge, O. & Valenta, A. (2013). Varierte representasjoner. *Tangenten*, *1*, 8-12.

Eriksen, D. (2022, 07.24.2022). *Lærere fortvilet over ny kunstig intelligens*. NRK. Hentet 29.04.2023 fra https://www.nrk.no/kultur/laerere-fortvilet-over-ny-kunstig-intelligens-1.16210580

Freiman, V. (2020). Types of Technology in Mathematics Education. I (s. 869-879). Springer International Publishing. https://doi.org/10.1007/978-3-030-15789-0_158

Gazoni, R. M. (2018). A Semiotic Analysis of Programming Languages. *Journal of Computer and Communications*, *06*(03), 91-101. https://doi.org/10.4236/jcc.2018.63007

Gleiss, M. S. & Sæther, E. (2021). *Forskningsmetode for lærerstudenter - Å utvikle ny kunnskap i forskning og praksis*. Cappelen Damm.

Godino, J. (1996). *MATHEMATICAL CONCEPTS, THEIR MEANINGS, AND UNDERSTANDING*. Conference of the International Group for the Psychology of Mathematics Education, Universitetet i Valencia.

Goldenberg, E. P., Carter, C. J., Mark, J., Reed, K., Spencer, D. & Coleman, K. (2021). Programming as Language and Manipulative for Second-Grade Mathematics. *Digital Experiences in Mathematics Education*, *7*(1), 48-65. https://doi.org/10.1007/s40751-020-00083-3

Goldin, G. A. (2020). Mathematical Representations. I S. Lerman (Red.), *Encyclopedia of Mathematics Education* (s. 566-572). Springer International Publishing. https://doi.org/10.1007/978-3-030-15789-0_103

Grønmo, S. (2016). *Samfunnsvitenskapelige metoder* (2. utg.). Fagbokforlaget.

Hana, G. M. (2013). *Matematiske byggesteiner*. Caspar Forlag.

Johnston-Wilder, S. & Pimm, D. (2004). Technology, mathematics and secondary schools: a brief, UK historical perspective.

Kleven, T. A. & Hjardemaal, F. R. (2018). *Innføring i pedagogisk forskningsmetode - En hjelp til kritisk tolkning og vurdering.* (3. utg.). Vigmostad & Bjørke AS.

Kristensen, T. E. & Kirfel, C. (2022). *Programmering i matematikkfaget*. Cappelen Damm.

Langdridge, D. (2006). *Psykologisk forskningsmetode - En innføring i kvalitative og kvanitative tilnærminger*. Tapir Akademinsk Forlag.

Lesh, R., Post, T. & Behr, M. (1987). Representations and Translations among Representations in Mathematics Learning and Problem Solving. *Problems of Representation in the Teaching and Learning of Mathematics, 21*, 33-40.

Liljedahl, P. (2021). *Building thinking classrooms in mathematics*. Corwin Press.

Loshin, P. (u.å.). *ASCII (American Standard Code for Information Interchange)*. TechTarget. Hentet 27.05.2023 fra https://www.techtarget.com/whatis/definition/ASCII-American-Standard-Code-for-Information-Interchange

Munthe, M. (2022a). Programmering i matematikklasserommet – Utfordringer elevene møter. *Acta Didactica Norden*, *16*(4). https://doi.org/10.5617/adno.9173

Munthe, M. (2022b). *Press 'Run' to Improve Mathematical Expertise (PRIME)* [Norges miljø- og biovitenskapelige universitet].

NCTM. (2014). *Principles to Actions: Ensuring Mathematical Success for All*. National Council of Teachers of Mathematics.

NESH. (2021). Forskningsetiske retningslinjer for samfunnsvitenskap og humaniora. https://www.forskningsetikk.no/retningslinjer/hum-sam/forskningsetiske-retningslinjer-for-samfunnsvitenskap-og-humaniora/

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

Sarama, J. & Clements, D. H. (2009). “Concrete” Computer Manipulatives in Mathematics Education. *Child Development Perspectives*, *3*(3), 145-150. https://doi.org/10.1111/j.1750-8606.2009.00095.x

Sarama, J. & Clements, D. H. (2016). Physical and Virtual Manipulatives: What Is “Concrete”? I (s. 71-93). Springer International Publishing. https://doi.org/10.1007/978-3-319-32718-1_4

Sevik, K. (2016). *Programmering i skolen*. Senter for IKT i utdanningen.

Sfard, A. (1991). On the dual nature of mathematical conceptions: Reflections on processes and objects as different sides of the same coin. *Educational Studies in Mathematics*, *22*(1), 1-36. https://doi.org/10.1007/BF00302715

Stephens, M. & Kadijevich, D. M. (2020). Computational/Algorithmic Thinking. I S. Lerman (Red.), *Encyclopedia of Mathematics Education* (s. 117-123). Springer International Publishing. https://doi.org/10.1007/978-3-030-15789-0_100044

Udir. (2019a, 18.11.2019). *Hva er kjerneelementer?* Utdanningsdirektoratet. https://www.udir.no/laring-og-trivsel/lareplanverket/stotte/hva-er-kjerneelementer/

Udir. (2019b, 27.03.2019). *Algoritmisk tenkning*. Utdanningsdirektoratet. https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/

Udir. (u.å.a). *Matematikk R (MAT03-02) - Kjerneelementer*. Utdanningsdirektoratet. Hentet 12.01.2023 fra https://www.udir.no/lk20/mat03-02/om-faget/kjerneelementer

Udir. (u.å.b). *Matematikk R (MAT03-02) - Kompetansemål og vurdering*. Utdanningsdirektoratet. https://www.udir.no/lk20/mat03-02/kompetansemaal-og-vurdering/kv294

UiB. (2023, 04.04.2023). *RETTE - UiBs prosjektoversikt*. Universitetet i Bergen. Hentet 05.05.2023 fra https://www.uib.no/personvern/128207/rette-uibs-prosjektoversikt#krav-til-registrering-i-rette

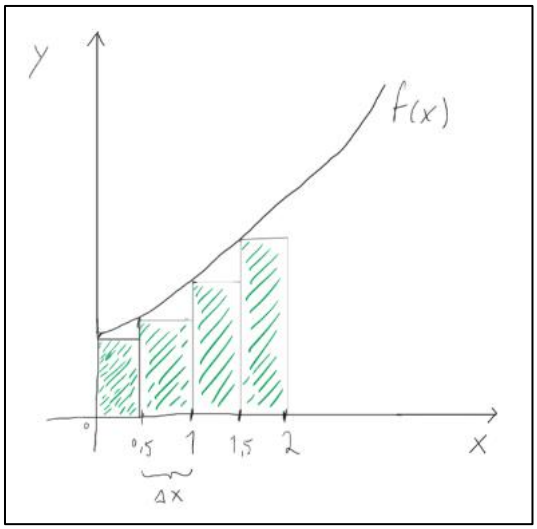
Wahlstrøm, C. (u.å.). *Programmering og algoritmisk tenkning*. Universitetet i Oslo. Hentet 12.01.2023 fra https://www.uv.uio.no/forskning/satsinger/fiks/kunnskapsbase/realfaglig-programmering/programmering-og-algoritmisk-tenkning-/

Wallace, C. S. (2004). Framing new research in science literacy and language use: Authenticity, multiple discourses, and the "Third Space". *Science Education*, *88*(6), 901-914. https://doi.org/10.1002/sce.20024

Vedlegg 1: Oppgave til elevene

OPPGAVE PÅ PAPIR (ca. 15 min):

Papiroppgave – planlegg koden



$$S_4 = f(0)\cdot\frac{1}{2} + f(0.5)\cdot\frac{1}{2} + f(1)\cdot\frac{1}{2} + f(1.5)\cdot\frac{1}{2}$$

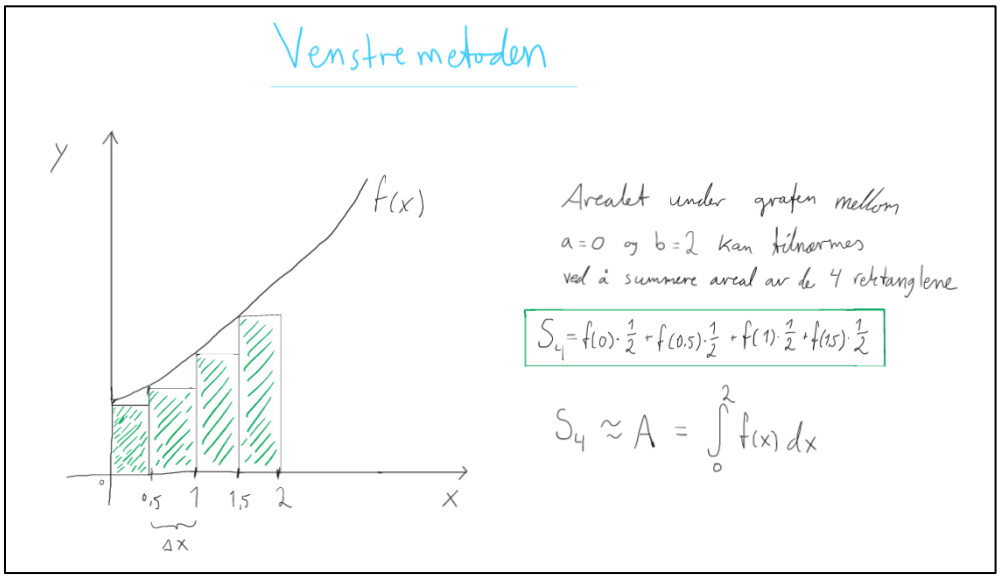
I eksempelet over gjorde vi en tilnærming av arealet under grafen mellom $a = 0$ og $b = 2$ ved å summere arealet av 4 rektangler.

Vi vil nå gjøre en ny tilnærming på samme intervall, men ved å bruke 50 rektangler. Dere skal snart skrive en kode som løser dette problemet, men først:

På papir: Skisser en idé til hvordan denne koden kan fungere.

OPPGAVE PÅ PC (ca. 1 time og 20 min):

Oppgave 1



På figuren over er $\Delta x = \frac{1}{2}$, og den representerer bredden til rektanglene. For å finne Δx må vi vite nedre grense (a), øvre grense (b) og antall rektangler (n).

Finn en formel for delta_x og bruk koden under til å printe verdien til delta_x når:

a) a = 0, b = 4 og n = 10

b) a = 2, b = 10 og n = 100

c) a = 0.5, b = 3.5 og n = 10000

Oppgave 2

Vi ser på følgende funksjon: $f(x) = \frac{1}{2}x^2 + 4$

Skriv en kode som bruker *venstremetoden* for å tilnærme en verdi for $\int_0^2 f(x)\,dx$ ved hjelp av:

a) 4 rektangler

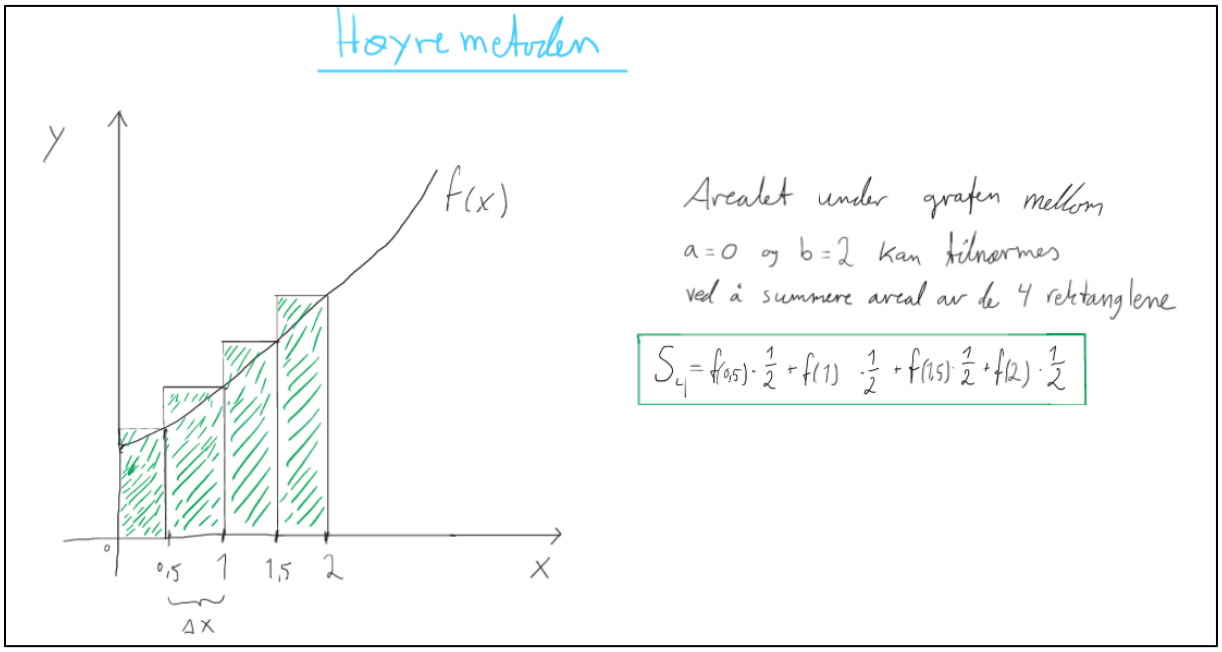
b) 50 rektangler

Det eksakte arealet under grafen i det gitte intervallet er $\int_0^2 \frac{1}{2}x^2 + 4\,dx = \frac{28}{3} \approx 9{,}333$.

c) Diskuter: Hvilken av tilnærmingene fra a) eller b) gir best tilnærming av arealet under grafen, og hvorfor?

d) Hvor mange rektangler trengs for å få en tilnærming med tre desimalers nøyaktighet?

Oppgave 3



Figuren over viser hvordan man kan tilnærme arealet under grafen ved å bruke *høyremetoden*.

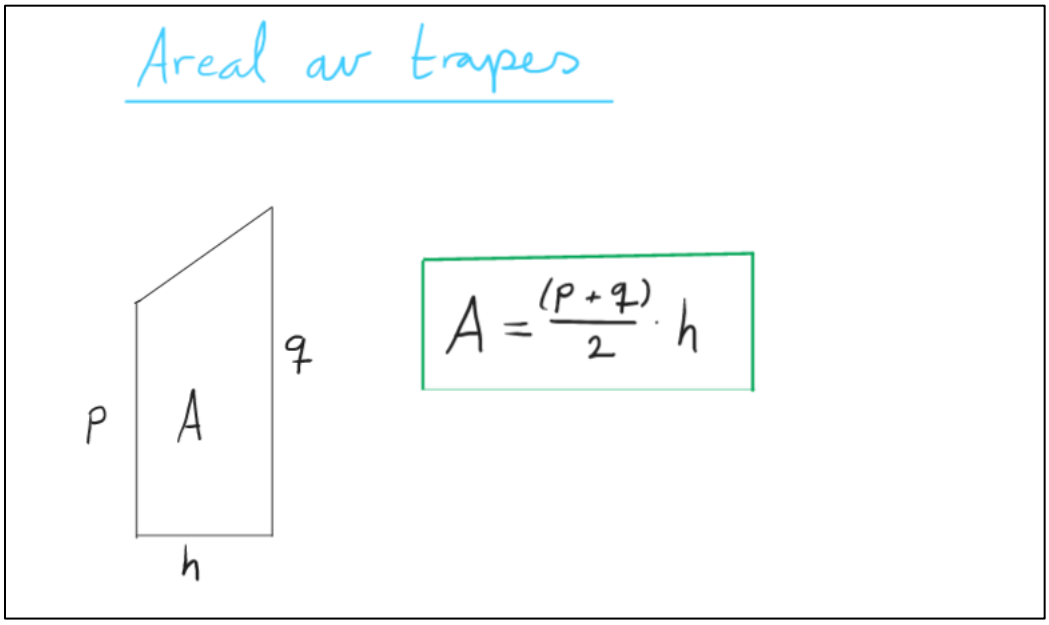
a) Skriv en kode som bruker *høyremetoden* for å gjøre en ny tilnærming av areal som i oppgave 2b.

b) Diskuter: Hva er forskjellene mellom resultatet i oppgave 2b og 3a, og hvorfor?

Oppgave 4

Vi kan også benytte oss trapeser for å tilnærme arealet under grafen, dette kaller vi *trapesmetoden*. Figuren under viser at formelen for arealet til et trapes er gitt ved:

$A = \frac{(p+q)}{2} \cdot h$



a) Skriv en kode som bruker trapesmetoden for å gjøre en ny tilnærming av areal som i oppgave 2b.

b) Hvor mange trapeser trengs for å få en tilnærming med tre desimalers nøyaktighet? Sammenlign dette resultatet med resultatet fra oppgave 2d.

Oppgave 5

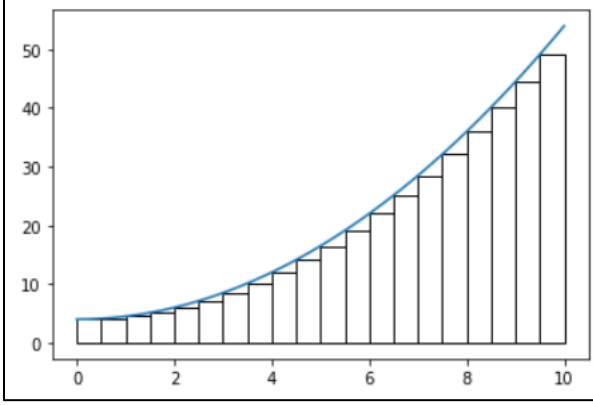
Kjør programmet under.

Diskuter: Hva som skjer når man øker antall rektangler?

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from matplotlib.patches import Rectangle as rekt

def f(x):
    return (1/2)*x**2+4
def plot(a,b,n):
    delta_x = (b-a)/n
    x_punkter_g = []
    y_punkter_g = []
    x_punkter = []
    y_punkter = []

    # Plotter rektangler
    for x in np.arange(a,b, ((b-a)/n)):
        x_punkter.append(x)
        y_punkter.append(f(x))
    fig = matplotlib.pyplot.figure()
    ax = fig.add_subplot()
    for i in x_punkter:
        ax.add_patch(rekt((i,0),float(delta_x),f(i),fill=False))  # Plotter venstre rektangler
    # Plotter graf
    for x in np.linspace(a,b,100):
        x_punkter_g.append(x)
        y_punkter_g.append(f(x))
    plt.plot(x_punkter_g,y_punkter_g)
    return None

a = 0
b = 10
n = int(input("Hvor mange rektangler vil du ha?"))
plot(a,b,n)
```

Eksempel på output:

```
Hvor mange rektangler vil du ha?20
```



Vedlegg 2: Samtykkeskjema

Vil du delta i forskningsprosjektet tilknyttet min masteroppgave om programmering og matematiske representasjoner?

Dette er et spørsmål til deg om å delta i et forskningsprosjekt hvor formålet er å undersøke hvordan programmering fungerer som en form for matematisk representasjon. I dette skrivet gir vi deg informasjon om målene for prosjektet og hva deltakelse vil innebære for deg.

Formål

Jeg studerer på Universitetet i Bergen for å bli lektor i matematikk og fysikk. Til våren skal jeg starte å skrive min masteroppgave innenfor matematikkdidaktikk. I oppgaven skal jeg undersøke hvordan programmering fungerer som en matematisk representasjonsform. Jeg er derfor på utkikk etter data om hvordan programmering representeres verbalt blant elever, og hvordan de arbeider sammen om en programmeringsoppgave.

Hvem er ansvarlig for forskningsprosjektet?

Matematisk institutt på Universitetet i Bergen er ansvarlig for prosjektet.

Sondre Waage Kolbeinsen er masterstudenten som gjennomfører prosjektet.

Hvorfor får du spørsmål om å delta?

Du blir spurt om å delta fordi du tar matematikk R2 som fag på videregående skole, og har vært innom programmeringsoppgaver tidligere.

Hva innebærer det for deg å delta?

Hvis du velger å delta i prosjektet, innebærer det at du arbeider med en matematikk-oppgave samtidig som det blir gjort skjerm-, lyd- og video-opptak (web-kamera) av gruppearbeidet. Oppgaven vil være innenfor matematikk, og alle opptak vil bli lagret elektronisk.

Hvis du ikke velger å delta vil du få et alternativt opplegg av lærer.

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.

Ditt personvern – hvordan vi oppbevarer og bruker dine opplysninger

Vi vil bare bruke opplysningene om deg til formålene vi har fortalt om i dette skrivet. Vi behandler opplysningene konfidensielt og i samsvar med personvernregelverket. Opplysningene vil bli lagret i en mappe som krever to-faktor autentisering.

Det er kun masterstudenten, Sondre Waage Kolbeinsen, som vil ha tilgang til opplysningene. De innsamlede dataene vil bli anonymisert under transkribering.

Hva skjer med personopplysningene dine når forskningsprosjektet avsluttes?

Prosjektet vil etter planen avsluttes 21. Juni 2023. Etter prosjektslutt vil datamaterialet med dine personopplysninger anonymiseres. Video-opptak vil ikke bli benyttet direkte i masteroppgaven, men som bakgrunnsmateriale for analyse.

Hva gir oss rett til å behandle personopplysninger om deg?

Vi behandler opplysninger om deg basert på ditt samtykke.

På oppdrag fra matematisk institutt har Personverntjenester vurdert at behandlingen av personopplysninger i dette prosjektet er i samsvar med personvernregelverket.

Dine rettigheter

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

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

Hvis du har spørsmål til studien, eller ønsker å vite mer om eller benytte deg av dine rettigheter, ta kontakt med:

- Masterstudent: Sondre Waage Kolbeinsen, mail: sko048@uib.no
- Matematisk institutt på Universitet i Bergen: Johan Lie, mail: Johan.Lie@uib.no

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

- Personverntjenester på epost (personverntjenester@sikt.no) eller på telefon: 53 21 15 00.

Med vennlig hilsen

Johan Lie	*Sondre Waage Kolbeinsen*
(Forsker/veileder)	(Masterstudent)

Samtykkeerklæring

Jeg har mottatt og forstått informasjon om masterprosjektet, og har fått anledning til å stille spørsmål. Jeg samtykker til:

- ☐ å delta i lyd- og video-opptak.
- ☐ Jeg samtykker til at mine opplysninger behandles frem til prosjektet er avsluttet

---

(Signert av prosjektdeltaker, dato)